PHP mt_rand()随机数安全

索马里的海贼 / 2017-10-10 09:07:00 / 浏览数 7869

前言

前段时间挖了不少跟mt_rand()相关的安全漏洞，基本上都是错误理解随机数用法导致的。这里又要提一下php官网manual的一个坑,看下关于mt_rand()的介绍:中文版[1] 英文版[2]，可以看到英文版多了一块黄色的Caution警告

> This function does not generate cryptographically secure values, and should not be used for cryptographic purposes. If you need a cryptographically secure value, consider using random_int(), random_bytes(), or openssl_random_pseudo_bytes() instead.

很多国内开发者估计都是看的中文版的介绍而在程序中使用了mt_rand()来生成安全令牌、核心加解密key等等导致严重的安全问题。

伪随机数

mt_rand()并不是一个真·随机数生成函数,实际上绝大多数编程语言中的随机数函数生成的都都是伪随机数。关于真随机数和伪随机数的区别这里不展开解释，只需要简单了解

> 伪随机是由可确定的函数（常用线性同余），通过一个种子（常用时钟），产生的伪随机数。这意味着：如果知道了种子，或者已经产生的随机数，都可能获得接下来随

简单假设一下 mt_rand()内部生成随机数的函数为:`rand = seed+(i*10)`其中`seed`是随机数种子，`i`是第几次调用这个随机数函数。当我们同时知道`i`和`rand`两个值的时候，就能很容易的算出seed的值来。比如`rand=21,i=2`代入函`21=seed+(2*10)`得到`seed=1`。是不是很简单，当我们拿到seed之后，就能计算出当`i`为任意值时候的`rand`的值了。

php的自动播种

从上一节我们已经知道每一次mt_rand()被调用都会根据seed和当前调用的次数i来计算出一个伪随机数。而且seed是自动播种的：

> Note: 自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 给随机数发生器播种 ，因为现在是由系统自动完成的。

那么问题就来了，到底系统自动完成播种是在什么时候，如果每次调用mt_rand()都会自动播种那么破解seed也就没意义了。关于这一点manual并没有给出详细信息。网上只能去翻源码[3]了：

```
PHPAPI void php_mt_srand(uint32_t seed)
{
   /* Seed the generator with a simple uint32 */
   php_mt_initialize(seed, BG(state));
   php_mt_reload();

   /* Seed only once */
   BG(mt_rand_is_seeded) = 1;
}
/* }}} */

/* {{{ php_mt_rand
*/
PHPAPI uint32_t php_mt_rand(void)
{
   /* Pull a 32-bit integer from the generator state
      Every other access function simply transforms the numbers extracted here */

   register uint32_t s1;

   if (UNEXPECTED(!BG(mt_rand_is_seeded))) {
       php_mt_srand(GENERATE_SEED());
   }

   if (BG(left) == 0) {
       php_mt_reload();
   }
   --BG(left);

   s1 = *BG(next)++;
   s1 ^= (s1 >> 11);
   s1 ^= (s1 <<  7) & 0x9d2c5680U;
   s1 ^= (s1 << 15) & 0xefc60000U;
   return ( s1 ^ (s1 >> 18) );
```

```
}
```

可以看到每次调用mt_rand()都会先检查是否已经播种。如果已经播种就直接产生随机数，否则调用php_mt_srand来播种。也就是说每个php cgi进程期间，只有第一次调用mt_rand()会自动播种。接下来都会根据这个第一次播种的种子来生成随机数。而php的几种运行模式中除了CGI(每个请求启动一个cgi进程，环境变量等导致效率低下，现在用的应该不多了)以外，基本都是一个进程处理完请求之后standby等待下一个，处理多个请求之后才会回收（超时也会回收）。
写个脚本测试一下

```
<?php
//pid.php
echo getmypid();
```

```
<?php
//test.php
$old_pid = file_get_contents('http://localhost/pid.php');
$i=1;
while(true){
   $i++;
   $pid = file_get_contents('http://localhost/pid.php');
   if($pid!=$old_pid){
       echo $i;
       break;
   }
}
```

测试结果：（windows+phpstudy）

apache 1000请求
nginx 500请求

当然这个测试仅仅确认了apache和nginx一个进程可以处理的请求数，再来验证一下刚才关于自动播种的结论：

```
<?php
//pid1.php
if(isset($_GET['rand'])){
   echo mt_rand();
}else{
   echo getmypid();
}
```

```
<?php
//pid2.php
echo mt_rand();
```

```
<?php
//test.php
$old_pid = file_get_contents('http://localhost/pid1.php');
echo "old_pid:{$old_pid}\r\n";
while(true){
   $pid = file_get_contents('http://localhost/pid1.php');
   if($pid!=$old_pid){
       echo "new_pid:{$pid}\r\n";
       for($i=0;$i<20;$i++){
           $random = mt_rand(1,2);
           echo file_get_contents("http://localhost/pid".$random.".php?rand=1")." ";
       }

       break;
   }
}
```

通过pid来判断,当新进程开始的时候，随机获取两个页面其中一个的 mt_rand() 的输出:

old_pid:972
new_pid:7752
1513334371 2014450250 1319669412 499559587 117728762 1465174656 1671827592 1703046841 464496438 1974338231 46646067
981271768 1070717272 571887250 922467166 606646473 134605134 857256637 1971727275 2104203195

拿第一个随机数1513334371去爆破种子：

smldhz@vm:~/php_mt_seed-3.2$ ./php_mt_seed 1513334371
Found 0, trying 704643072 - 738197503, speed 28562751 seeds per second

seed = 735487048
Found 1, trying 1308622848 - 1342177279, speed 28824291 seeds per second
seed = 1337331453
Found 2, trying 3254779904 - 3288334335, speed 28811010 seeds per second
seed = 3283082581
Found 3, trying 4261412864 - 4294967295, speed 28677071 seeds per second
Found 3

爆破出了3个可能的种子，数量很少 手动一个一个测试:

```
<?php
mt_srand(735487048);//■■■■
for($i=0;$i<21;$i++){
   echo mt_rand()." ";
}
```

输出:

1513334371 2014450250 1319669412 499559587 117728762 1465174656 1671827592 1703046841 464496438 1974338231 46646067
981271768 1070717272 571887250 922467166 606646473 134605134 857256637 1971727275 2104203195 1515656265

前20位跟上面脚本获取的一模一样，确认种子就是1513334371。有了种子我们就能计算出任意次数调用mt_rand()生成的随机数了。比如这个脚本我生成了21位，最后一位
如果跑完刚才的脚本之后没访问过站点，那么打开http://localhost/pid2.php就能看到相同的1515656265。
所以我们得到结论：

php的自动播种发生在php
cgi进程中第一次调用mt_rand()的时候。跟访问的页面无关，只要是同一个进程处理的请求，都会共享同一个最初自动播种的种子。

php_mt_seed

我们已经知道随机数的生成是依赖特定的函数，上面曾经假设为`rand = seed+(i*10)`。对于这样一个简单的函数，我们当然可以直接计算（口算）出一个（组）解来，但 mt_rand()
实际使用的函数可是相当复杂且无法逆运算的。有效的破解方法其实是穷举所有的种子并根据种子生成随机数序列再跟已知的随机数序列做比对来验证种子是否正确。php_
MAX输出的种子（下面实例中有用到）。

安全问题

说了这么多，那到底随机数怎么不安全了呢？其实函数本身没有问题，官方也明确提示了生成的随机数不应用于安全加密用途（虽然中文版本manual没写）。问题在于开发

夜深人静，等待apache(nginx)收回所有php进程（确保下次访问会重新播种），访问一次验证码页面，根据验证码字符逆推出随机数，再根据随机数爆破出随机数种子。

实例

1. PHPCMS MT_RAND SEED CRACK致authkey泄露 雨牛写的比我好，看他的就够了
2. Discuz x3.2 authkey泄露 这个其实也差不多。官方已出补丁，有兴趣的可以自己去分析一下。

---

http://php.net/manual/zh/function.mt-rand.php

http://php.net/manual/en/function.mt-rand.php

https://github.com/php/php-src/blob/e23da65550680230618bc26fb34d19baa89157fe/ext/standard/mt_rand.c

http://www.openwall.com/php_mt_seed/

上一篇：Web Hacking 101 中文版 下一篇：PHPCMS MT_RAND SE...

1. 3 条回复


hades 2017-10-10 15:00:26

嗯 很丑~

0 回复Ta


0r3ak 2017-10-10 15:40:47

其实还可以参考这篇文章：http://wonderkun.cc/index.html/?p=585，随机数之前也是ctf的常见姿势

0 回复Ta


hades 2017-10-10 15:53:53

作者也在那个群里地~~

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板