Phar与Stream Wrapper造成PHP RCE的深入挖掘

zsx / 2018-10-22 22:21:56 / 浏览数 2965 安全技术 WEB安全 顶(3) 踩(0)

Blog： https://blog.zsxsoft.com/post/38

今年的HITCON打完了，沉迷写前端搞Nextjs骚操作的我成功爆0（雾），不想写前端了.jpg。

先跑个题。
HITCON 2016上，orange 出了一道PHP反序列化。
HITCON 2017上，orange 出了一道Phar + PHP反序列化。
HITCON 2018上，orange 出了一道file_get_contents + Phar + PHP反序列化。
让我们期待HITCON 2019的操作（雾。

Phar RCE

今年HITCON上，baby cake这一题，涉及到了今年BlackHat大会上的Sam Thomas分享的File Operation Induced Unserialization via the “phar://” Stream Wrapper这个议题，见：https://i.blackhat.com/us-18/Thu-August-9/us-18-Thomas-Its-A-PHP-Unserialization-Vulnerability-Jim-But-Not-As-We-Know-It-wp.pd 。它的主要内容是，通过phar://协议对一个phar文件进行文件操作，如file_get_contents，就可以触发反序列化，从而达成RCE的效果。

在文章开头部分，让我先对phar反序列化做一些小小的分析。我们直接阅读PHP源码。在 phar.c#L618 处，其调用了php_var_unserialize。

```
if (!php_var_unserialize(metadata, &p, p + zip_metadata_len, &var_hash)) {
```

因此可以构造一个特殊的phar包，使得攻击代码能够被反序列化，从而构造一个POP链。这一部分已经太常见了，CTF比赛中都出烂了，没什么值得继续讨论的。值得关注的

Stream API

因此，为解决这个问题，我们需要首先阅读此函数的源码。大概在此处：https://github.com/php/php-src/blob/PHP-7.2.11/ext/standard/file.c#L548 ，重点关注此行：

```
stream = php_stream_open_wrapper_ex(filename, "rb",
               (use_include_path ? USE_PATH : 0) | REPORT_ERRORS,
               NULL, context);
```

可以注意，其使用的是php_stream系列API来打开一个文件。阅读PHP的这篇文档：Streams API for PHP Extension Authors，可知，Stream API是PHP中一种统一的处理文件的方法，并且其被设计为可扩展的，允许任意扩展作者使用。而本次事件的主角，也就是phar这个扩展，其就注册了phar://这个stream wrapper。可以使用stream_get_wrapper看到系统内注册了哪一些wrapper，但其余的没什么值得关注的。

```
php > var_dump(stream_get_wrappers());
array(12) {
 [0]=>
 string(5) "https"
 [1]=>
 string(4) "ftps"
 [2]=>
 string(13) "compress.zlib"
 [3]=>
 string(14) "compress.bzip2"
 [4]=>
 string(3) "php"
 [5]=>
 string(4) "file"
 [6]=>
 string(4) "glob"
 [7]=>
 string(4) "data"
 [8]=>
 string(4) "http"
 [9]=>
 string(3) "ftp"
 [10]=>
 string(4) "phar"
 [11]=>
 string(3) "zip"
}
```

那么，注册一个 stream
wrapper，能实现什么功能呢？很容易就能找到其定义：https://github.com/php/php-src/blob/8d3f8ca12a0b00f2a74a27424790222536235502/main/php_streams

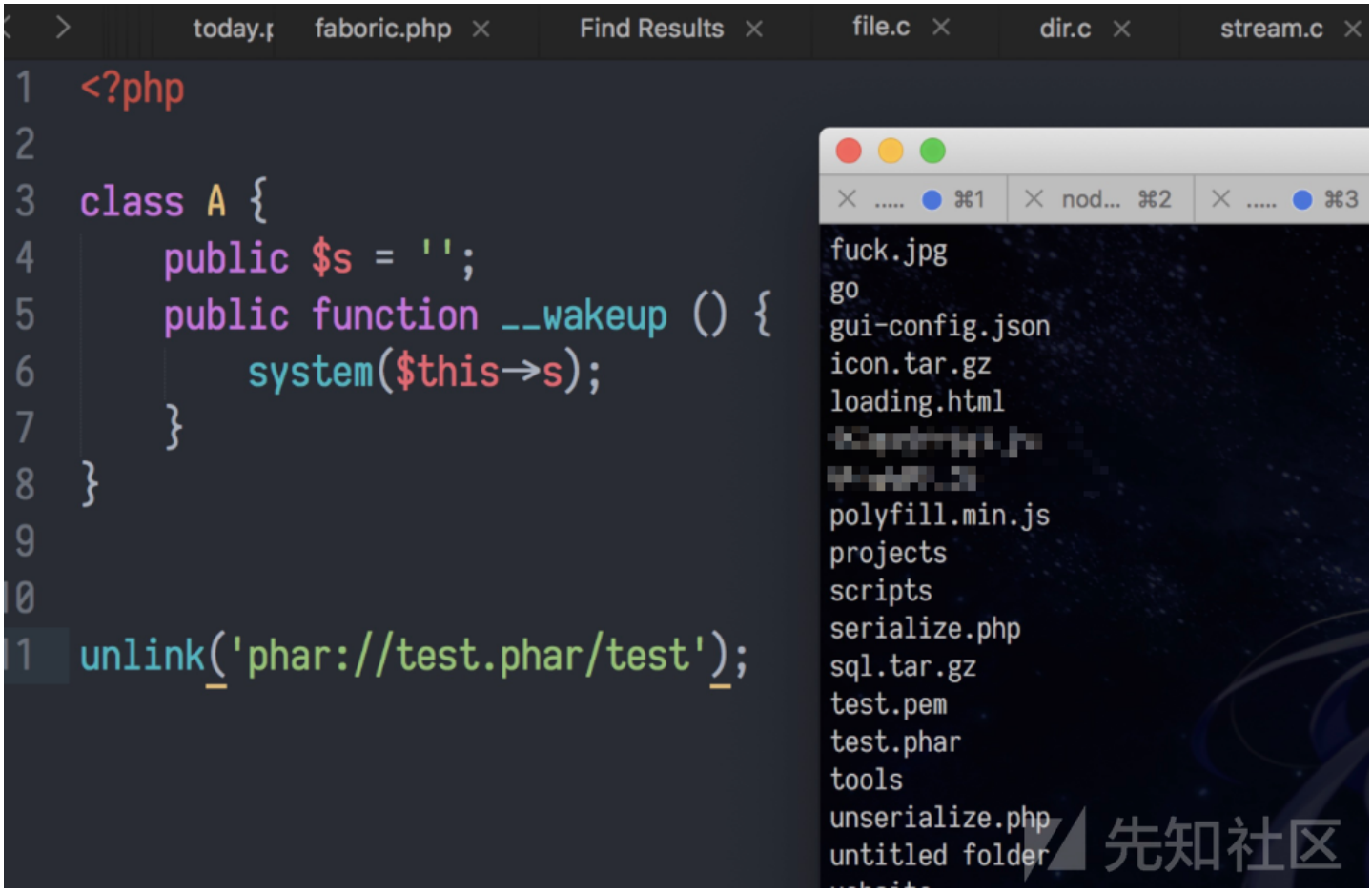
```
typedef struct _php_stream_wrapper_ops {
    /* open/create a wrapped stream */
    php_stream *(*stream_opener)(php_stream_wrapper *wrapper, const char *filename, const char *mode,
            int options, zend_string **opened_path, php_stream_context *context STREAMS_DC);
    /* close/destroy a wrapped stream */
    int (*stream_closer)(php_stream_wrapper *wrapper, php_stream *stream);
    /* stat a wrapped stream */
    int (*stream_stat)(php_stream_wrapper *wrapper, php_stream *stream, php_stream_statbuf *ssb);
    /* stat a URL */
    int (*url_stat)(php_stream_wrapper *wrapper, const char *url, int flags, php_stream_statbuf *ssb, php_stream_context *conte
    /* open a "directory" stream */
    php_stream *(*dir_opener)(php_stream_wrapper *wrapper, const char *filename, const char *mode,
            int options, zend_string **opened_path, php_stream_context *context STREAMS_DC);
    const char *label;
    /* delete a file */
    int (*unlink)(php_stream_wrapper *wrapper, const char *url, int options, php_stream_context *context);
    /* rename a file */
    int (*rename)(php_stream_wrapper *wrapper, const char *url_from, const char *url_to, int options, php_stream_context *conte
    /* Create/Remove directory */
    int (*stream_mkdir)(php_stream_wrapper *wrapper, const char *url, int mode, int options, php_stream_context *context);
    int (*stream_rmdir)(php_stream_wrapper *wrapper, const char *url, int options, php_stream_context *context);
    /* Metadata handling */
    int (*stream_metadata)(php_stream_wrapper *wrapper, const char *url, int options, void *value, php_stream_context *context)
} php_stream_wrapper_ops;
```

因此，我们发现，一个 stream
wrapper，它支持以下功能：打开文件（夹）、删除文件（夹）、重命名文件（夹），以及获取文件的meta。我们很容易就能断定，类似unlink等函数也是同样通过这个 streams api 进行操作。

Sam Thomas 的 pdf 指出

> This is true for both direct file operations (such as
> "file_exists") and indirect operations such as those that occur during external entity processing
> within XML (i.e. when an XXE vulnerability is being exploited).

我们通过试验也很容易发现，类似unlink等函数也均是可以使用的。



知道创宇404实验室的研究员 seaii 更为我们指出了所有文件函数均可使用（ https://paper.seebug.org/680/ ）：

- `fileatime` / `filectime` / `filemtime`
- `stat` / `fileinode` / `fileowner` / `filegroup` / `fileperms`
- `file` / `file_get_contents` / `readfile` / `fopen``
- `file_exists` / `is_dir` / `is_executable` / `is_file` / `is_link` / `is_readable` / `is_writeable` / `is_writable`
- `parse_ini_file`
- `unlink`
- `copy`

受影响函数列表			
fileatime	filectime	file_exists	file_get_contents
file_put_contents	file	filegroup	fopen
fileinode	filemtime	fileowner	fileperms
is_dir	is_executable	is_file	is_link
is_readable	is_writable	is_writeable	parse_ini_file
copy	unlink	stat	readfile

仅仅是知道一些受影响的函数，就够了吗？为什么就可以使用了呢？

寻找受害者

当然不够。我们需要先找到其原理，然后往下深入挖掘。
先看`file_get_contents`的代码。其调用了

```
stream = php_stream_open_wrapper_ex(filename, "rb" ....);
```

这么个函数。

再看unlink的代码，其调用了

```
wrapper = php_stream_locate_url_wrapper(filename, NULL, 0);
```

这么个函数。

从php_stream_open_wrapper_ex的实现，可以看到，其也调用了php_stream_locate_url_wrapper。这个函数的作用是通过url来找到对应的wrapper。我们可以看到，phar组件注册了phar://这个wrapper，
https://github.com/php/php-src/blob/67b4c3379a1c7f8a34522972c9cb3adf3776bc4a/ext/phar/stream.c
其定义如下：

```
const php_stream_wrapper_ops phar_stream_wops = {
   phar_wrapper_open_url,
   NULL,                  /* phar_wrapper_close */
   NULL,                  /* phar_wrapper_stat, */
   phar_wrapper_stat,     /* stat_url */
   phar_wrapper_open_dir, /* opendir */
   "phar",
   phar_wrapper_unlink,   /* unlink */
   phar_wrapper_rename,   /* rename */
   phar_wrapper_mkdir,    /* create directory */
   phar_wrapper_rmdir,    /* remove directory */
   NULL
};
```

接着，让我们翻这几个函数的实现，会发现它们都调用了phar_parse_url，这个函数再调用phar_open_or_create_filename -> phar_create_or_parse_filename -> phar_open_from_fp -> phar_parse_pharfile -> phar_parse_metadata -> phar_var_unserialize。因此，明面上来看，所有文件函数，均可以触发此phar漏洞，因为它们都直接或间接地调用了这个wrapper。

只是这些文件函数，就够了吗？
当然不够。这是一个所有的和IO有关的函数，都可能触发的问题。

前面我已经指出，它们都有一个共同特征，就是调用了php_stream_locate_url_wrapper。但是这个不那么好用，换php_stream_open_wrapper更合适点。让我们找

我们很快就能发现，操作文件的touch，也是能触发它的。不看文件了，我们假设文件全部都能用。

我们会惊讶（一点都不）地发现：

exif

- exif_thumbnail
- exif_imagetype

gd

- imageloadfont
- imagecreatefrom***

hash

- hash_hmac_file
- hash_file
- hash_update_file
- md5_file
- sha1_file

file / url

- get_meta_tags
- get_headers

standard

- getimagesize
- getimagesizefromstring

zip

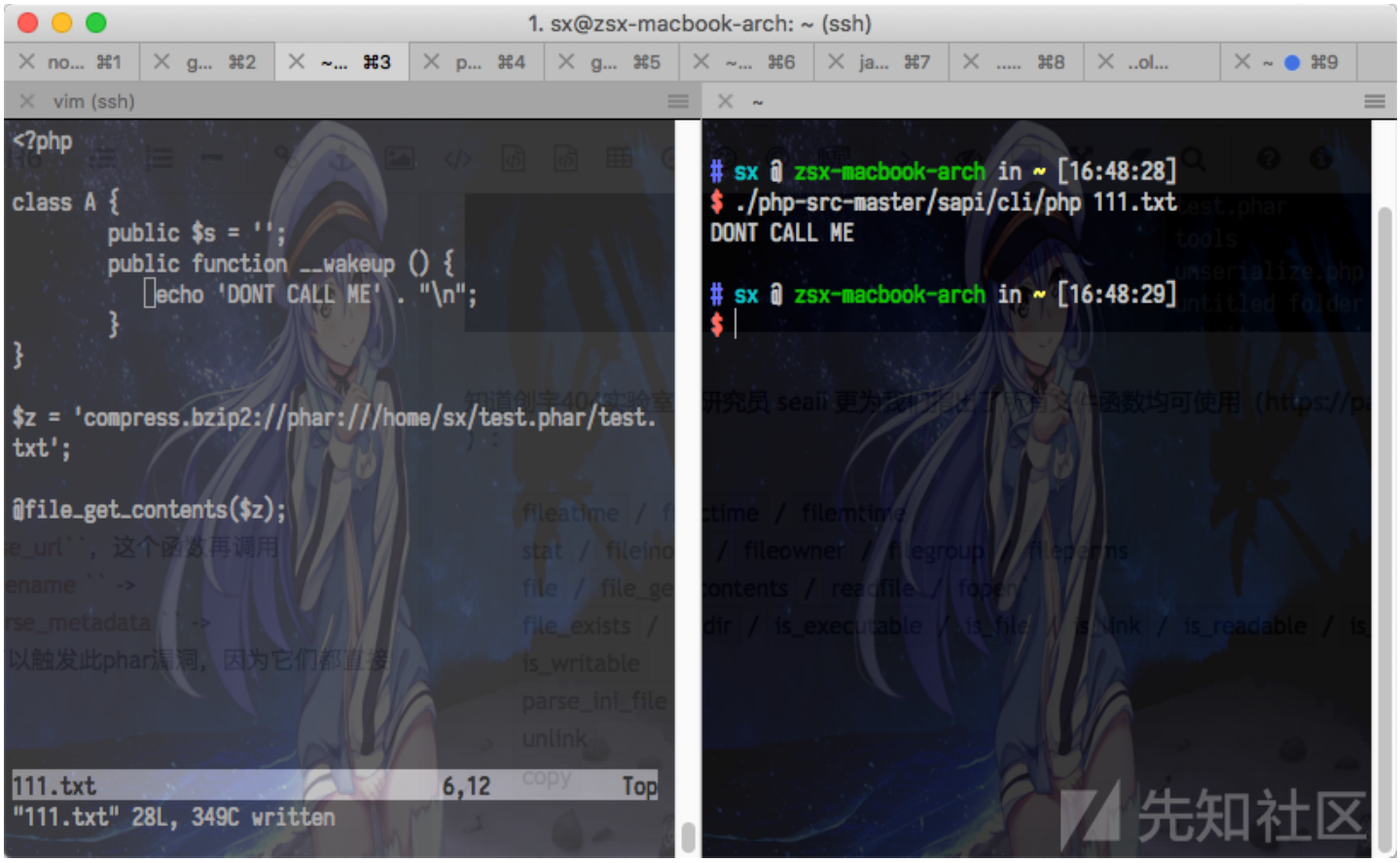
```
$zip = new ZipArchive();
$res = $zip->open('c.zip');
$zip->extractTo('phar://test.phar/test');
```

Bzip / Gzip

这，够了吗？non non哒哟！

如果题目限制了，phar://不能出现在头几个字符怎么办？请欣赏你船未见过的船新操作。

```
$z = 'compress.bzip2://phar:///home/sx/test.phar/test.txt';
```



当然，它同样适用于compress.zlib://。

Postgres

再来个数据库吧！

```
<?php
$pdo = new PDO(sprintf("pgsql:host=%s;dbname=%s;user=%s;password=%s", "127.0.0.1", "postgres", "sx", "123456"));
@$pdo->pgsqlCopyFromFile('aa', 'phar://test.phar/aa');
```

当然，pgsqlCopyToFile和pg_trace同样也是能使用的，只是它们需要开启phar的写功能。

MySQL

还有什么骚操作呢？
……MySQL？
走你！

我们注意到，LOAD DATA LOCAL INFILE也会触发这个php_stream_open_wrapper. 让我们测试一下。

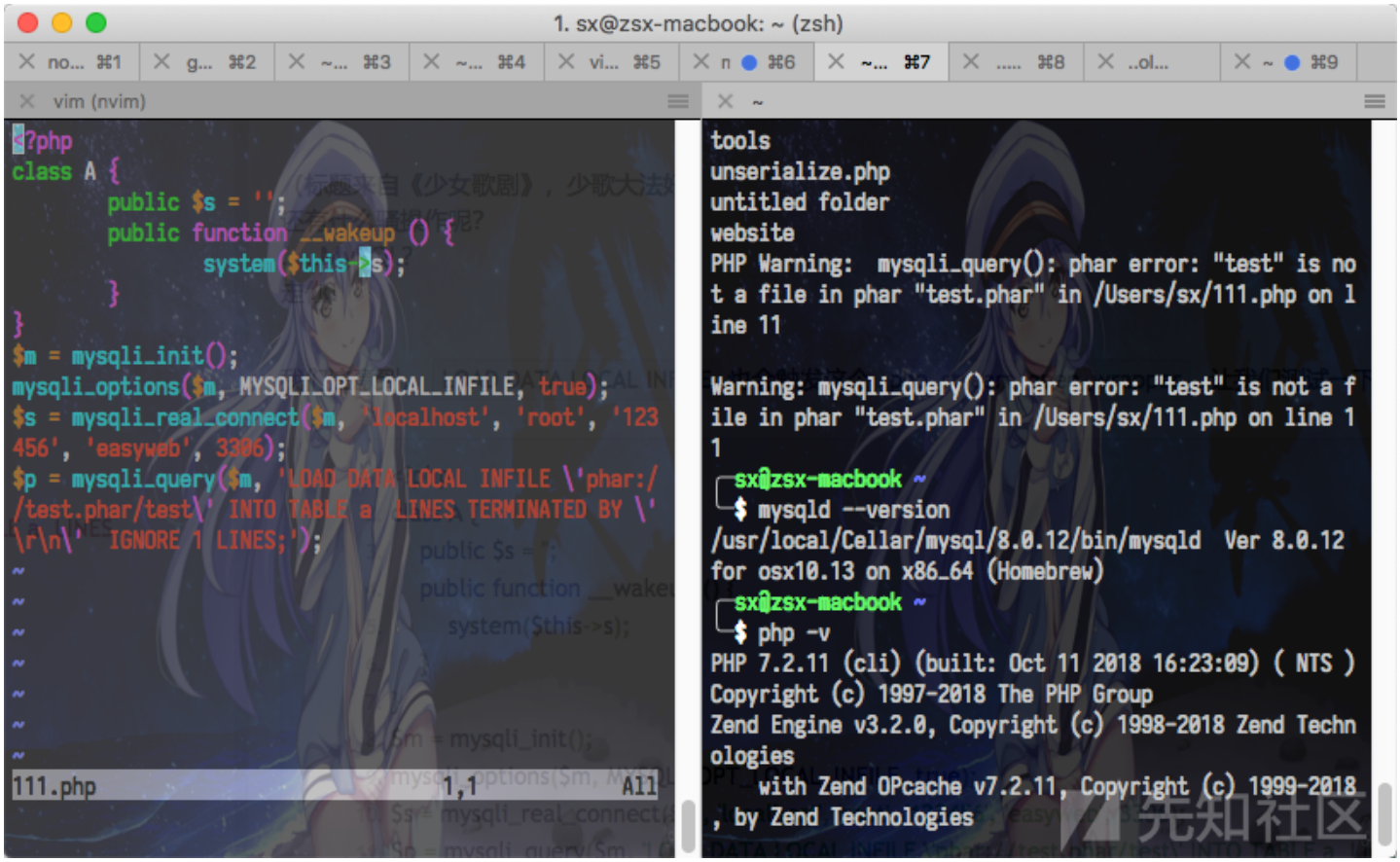
```
<?php
class A {
   public $s = '';
   public function __wakeup () {
       system($this->s);
   }
}
$m = mysqli_init();
mysqli_options($m, MYSQLI_OPT_LOCAL_INFILE, true);
```

```
$s = mysqli_real_connect($m, 'localhost', 'root', '123456', 'easyweb', 3306);
$p = mysqli_query($m, 'LOAD DATA LOCAL INFILE \'phar://test.phar/test\' INTO TABLE a  LINES TERMINATED BY \'\r\n\'  IGNORE 1 L
```

再配置一下mysqld。

```
[mysqld]
local-infile=1
secure_file_priv=""
```

.....然后，走你！



这就是我想要看到的舞台！——长颈鹿
很可惜，这不是默认配置；但是，嗯，很有意思。

我相信，PHP代码内部还有相当多的`php_stream_open_wrapper`等待挖掘，这只是关于stream wrapper利用的一小步。

点击收藏 | 11 关注 | 4

上一篇：带外通道(OOB)技术清单 下一篇：2018开源静态分析工具-第一部分...

1. 2 条回复

niexinming 2018-10-22 22:32:35

马克

0 回复Ta

kingkk 2018-10-24 08:52:22

赞！

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