

# XDCTF Writeup 天枢

## MISC 100

这里不得不说主办方的脑洞很大，对着 B 平面撸了半天，发现图像最开始的一段被篡改成了 0xf6-0xf8、0xfb-0xff，但是不知道代表什么。后来主办方提示 braintools，去 github 上找到了：<https://github.com/mbikovitsky/BrainTools>

编译了一下，然后运行：

```
F:\Hades\CTF\xdctf2015\BrainTools-master\BrainTools-master\BrainTools\bin\Release>bftools decode braincopter F:\Hades\CTF\xdctf2015\zzzzzzzyu.png -o 1.txt
F:\Hades\CTF\xdctf2015\BrainTools-master\BrainTools-master\BrainTools\bin\Release>bftools run 1.txt
XDCTF{ji910-dad9jq0-iopuno}
```

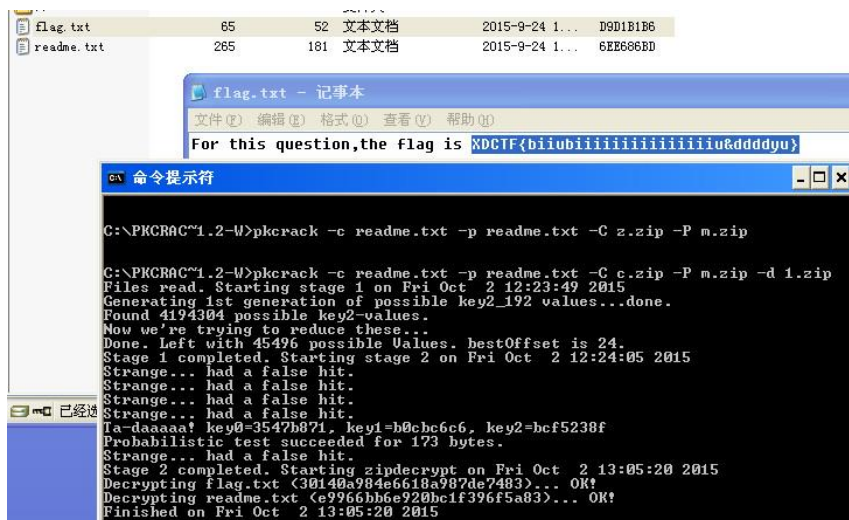
本题 Flag: XDCTF{ji910-dad9jq0-iopuno}

## MISC 200

首先利用 foremost 进行提取，得到两个 zip 包：



最开始以为都是伪加密，去了密码之后显示 CRC 校验错误，看来是有密码的。通过 readme.txt 解压之后的大小，判断是同一个文件。通过 PKCrack 进行明文密码攻击：



得到 flag: XDCTF{biubiiiiiiiiiiiiiu&ddddyu}

# PWN 100

这里给的是一个 word2007 rtf 文件的一个栈溢出漏洞样本，通过样本关键字可以搜索到 POC 相关分析的文档和漏洞号：

<http://www.2cto.com/Article/201301/185786.html>

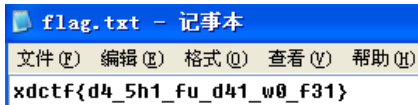
在虚拟机里面安装对应版本的 office，先用 windbg 搜索 90909090，定位到 shellcode 的位置。然用 OD 挂载进程，可以在内存中看到 shellcode 的情况：

00124C64	90	nop	
00124C65	90	nop	
00124C66	90	nop	
00124C67	90	nop	
00124C68	90	nop	
00124C69	90	nop	
00124C6A	90	nop	
00124C6B	90	nop	
00124C6C	4A	dec	edx
00124C6D	D6	salc	
00124C6E	90	nop	
00124C6F	96	xchg	eax, esi
00124C70	9A	xchgb	eax, esi

00124DE9	8B 34 8B 01 D6 31 FF AC C1 CF 00 01 C7 38 E0 75	???	????? 按部
00124DF9	F6 03 7D F8 38 7D 24 75 EA 58 88 58 24 01 D3 66	? ? ? ? ?	????? 已读第 0 段
00124E09	8B 0C 4B 8B 58 1C 01 D3 84 0A 88 01 D0 89 44 24	?K 按部 上 按部 下	????? 按部 上 按部 下
00124E19	24 58 5B 61 59 58 51 FF E0 5F 5F 58 88 12 EB 80	\$[ [avZQ] 按部 2? 壳	????? \$[ [avZQ] 按部 2? 壳
00124E29	50 6A 01 8D 85 B2 00 00 50 58 67 31 8B 6F 87 FF	JJ 按部 . . Ph? 堆?	????? JJ 按部 . . Ph? 堆?
00124E39	D5 8B FE E5 A2 56 68 95 8D 90 FF D5 C3 06 7C	栈 码 h 结? 堆?	????? 栈 码 h 结? 堆?
00124E49	0A 8B FB E0 75 0B 47 12 72 6F 6B 00 53 FF 05	h u 置 broj. 的 句	????? h u 置 broj. 的 句
00124E59	63 6D 6A 2E 65 78 20 2F 63 20 65 63 68 6F 20	cnd.exe c echo	????? cnd.exe c echo
00124E69	78 64 34 75 76 6B 7A 64 5F 35 35 68 31 5F 65 75 5F	xdcxtcfq4 5h1 fu	????? xdcxtcfq4 5h1 fu
00124E79	64 34 31 5F 77 38 5F 66 33 31 70 2D 30 2C 63 38	d41_u0_f31) : c;	????? d41_u0_f31) : c;
00124E89	52 66 6C 61 67 2E 7A 78 78 7A 26 26 20 61 74 74	\flag.txt && att	????? \flag.txt && att
00124E99	7C 69 62 20 63 5A 66 66 61 67 2E 7A 78 74 20	rib c:\flag.txt	????? rib c:\flag.txt

得到最终 flag:



# Reverse 100

这题有个假的函数验证函数 sub\_4008E1:

```

1 if ( (unsigned int)sub_400626(0x601328LL, 0xCu) == 1 )
2 {
3     qword_601338 = aCongratulation;
4     for ( i = 0; i < strlen((const char *)&good_crack_base64); ++i )
5         byte_601310[(signed __int64)i] = pwd[i
6             - 12
7             * ((unsigned __int64)((unsigned __int128)(0x0AAAAAAAAAAAAABLL * i) >> 64) >> 3)] ^ L
8
9     sub_4006D5(byte_601310, 24, 12);
10    for ( j = 0; j <= 0x17; ++j )
11    {
12        if ( byte_601310[(signed __int64)(signed int)j] <= 31 )
13            byte_601310[(signed __int64)(signed int)j] += 32;
14    }
15    v1 = 1;
16    for ( k = 0; k <= 0x17; ++k )
17    {
18        if ( byte_601310[(signed __int64)(signed int)k] != byte_6012A0[(signed __int64)(signed int)k] )
19            v1 = 0;
20    }
21 }

```

基本算法逻辑是异或上一段 base64 然后再异或 0x07，解出来之后一直不对，后来发现真正的逻辑在 sub\_400787 处，唯一的区别是少异或了一个 0x07，写脚本暴力跑一下：

```

for ii in range(0, a2/2):
    v3 = result[ii]
    result[ii] = result[a2 - ii - a3/2 -1]
    result[a2 - ii - a3/2 -1] = v3

print result

for ii in range(0, 24):
    if result[ii] > 31:
        s1.append(result[ii]-32)
    else:
        s1.append(result[ii])

for ii in range(0, 24):
    r1 = chr(result[ii] ^ ord(a[ii]))
    r2 = chr(s1[ii] ^ ord(a[ii]))

    print hex(ord(r1)), r1, hex(ord(r2)), r2

```

```

0x75 u 0x55 U
0x27 ' 0x47 G
0x72 r 0x52 R
0x45 E 0x65 e
0x7f 0x5f _
0x61 a 0x41 A
0x77 w 0x17 □
0x65 e 0x45 E
0x53 S 0x73 s
0x30 0 0x50 P
0x6d m 0x4d M
0x45 E 0x65 e
0x75 u 0x55 U
0x7 0x27 '
0x72 r 0x52 R
0x45 E 0x65 e
0x7f 0x5f _
0x41 A 0x61 a
0x77 w 0x57 W
0x45 E 0x65 e
0x53 S 0x73 s
0x10 0x30 0
0x6d m 0xd
0x65 e 0x5 □

```

根据主办方的提示，都是小写字母，所以最后的 flag: XDCTF{u’re\_awes0me}

## Reverse 200

这题是在入口点之后，会有一段代码解密的模块，对 0x401166 模块的代码进行解码，需要在解密模块之后分析代码功能，还有一堆花指令。

对 0x401166 下硬件访问断点，然后会断在这里：

00401056	. BA 60114000	mov	edx, 00401160	入口地址	EDX 00401160 cr
00401058	> 3E28032 88	xor	byte ptr [edx], 88		EBX 7FFD9000
00401059	. 42	inc	edx	crack2.00401160	ESP 0012FF6C
00401060	. 81FA E0114000	cmp	edx, 004011E0		EBP 0012FF00
00401066	. 75 F3	jnz	short 00401658		ESI 00000000
00401068	. 90	nop			EDI 7C930208 nt
00401069	. 90	nop			EIP 0040165F cr
0040106A	. 90	nop			C 0 ES 0023 32
0040106B	. 90	nop			P 1 CS 001B 32
0040106C	. 61	popad			A 0 SS 0023 32
0040106D	. 56	push	esi		Z 0 DS 0023 32
0040106E	. 56	push	esi		S 0 FS 003B 32
0040106F	. 6A 01	push	1		T 0 GS 0000 NU
00401671	. BA 121B4000	mov	edx, 00401B12		D 0
00401676	. FFE2	jmp	edx		O 0 LastErr ER
00401678	> 5F	pop	edi		EFL 00000206 (N
00401679	. 5E	pop	esi		MM0 0105 0104 0
0040167A	. 5B	pop	ebx		MM1 0069 006E 0
0040167B	. 8BE5	mov	esp, ebp		MM2 0000 0000 0
0040167C	. 8BE5	mov	esp, ebp		MM3 0000 0000 0
0040167D	. 8BE5	mov	esp, ebp		MM4 0000 0000 0

跳过这个循环，然后看看 00401160 处的代码：

00401160	\$ 55	push	ebp		
00401161	. 8BEC	mov	ebp, esp		
00401163	. 81EC 50010000	sub	esp, 150		
00401169	. 53	push	ebx		
0040116A	. 56	push	esi		
0040116B	. 57	push	edi		
0040116C	. 58	push	eax		
00401171	. E8 0B080000	call	0040C0AC		ASCII "Give me the flag:"
00401176	. 83C4 04	add	esp, 4		
00401179	. 81EA 2E080000	sub	edx, 82E		
0040117F	. 0085 F0FEFFFF	lea	eax, dword ptr [ebp-110]		
00401185	. 50	push	eax		
00401186	. E8 83070000	call	0040190E		
0040118B	. 83C4 04	add	esp, 4		

这样就看到了核心代码，往下跟一跟输入：

004011E8	83BD E8FEFF	cmp	dword ptr [ebp-118], 19	25
004011F2	7D 09	jge	short 004011FD	
004011F4	83BD E8FEFF	cmp	dword ptr [ebp-118], 0	
004011FB	7F 1A	jg	short 00401217	
004011FD	68 60CA0000	push	0040CA60	
00401202	E8 F4070000	call	00401001	ASCII "You shall not pass",LF

长度要要求小于 25，并且以 XDCTF{开头：

```

004012DA . 8B95 D0FEFFFI mov     edx, dword ptr [ebp-130]
004012E0 . 0FBE8415 F0FI movsx   eax, byte ptr [ebp+edx-110]
004012E8 . 3BC8      cmp     ecx, eax
004012EA . 74 1A     je      short 00401306
004012EC . 68 7CA400 push   0040CA7C
004012F1 . E8 0B070000 call   00401A01
004012F4 . 83C4 04   add     esp, 4

```

接下来会判断第 24 个字符是不是}，并且指出 和\$的位置：

```

0040130F . 83F9 7D    cmp     ecx, 7D
00401312 . 74 1A      je      short 0040132E
00401314 . 68 7CCA4000 push    0040CA7C
00401319 . E8 E3060000 call    00401A01
0040131E . 83C4 04     add     esp, 4
00401321 . E8 A5070000 call    00401ACB
00401326 . 83C8 FF     or      eax, FFFFFFFF
00401329 . E9 4A030000 jmp      00401678
0040132E > 0FB95 FCFF movsx   edx, byte ptr [ebp-104]
00401335 . 83FA 5F     cmp     edx, 5F
00401338 . 75 0C      jnz     short 00401346
0040133A . 0FB85 02FFF movsx   eax, byte ptr [ebp-FE]
00401341 . 83F8 24     cmp     eax, 24

```

所以使用 `XDCTF{123456 7890a$bcde}` 进行测试，找到第一部分的 check 代码：

[illegible]

循环相减 XDCTF , 等于固定的数值, 推出第一部分 Congra, 接下来是第二部分的 check:

004010E0	-	8D55 F8	lea	edx, dword ptr [ebp+8]
004010F1	-	8955 F4	mov	dword ptr [ebp+C], edx
004010F4	-	8B45 08	mov	eax, dword ptr [ebp+8]
004010F7	-	8945 F0	mov	dword ptr [ebp-10], eax
004010FA	>	8B4D F0	mov	ecx, dword ptr [ebp-10]
004010FD	-	8A11	mov	dl, byte ptr [ecx]
004010FF	-	8B55 EF	mov	byte ptr [ebp-11], dl
00401102	-	8B45 F4	mov	eax, dword ptr [ebp-C]
00401105	-	3A10	cmp	dl, byte ptr [eax]
00401107	-	75 2E	jnz	short 00401137
00401109	-	8D7D FE 00	cmp	byte ptr [ebp-11], 0
0040110D	-	74 1F	je	short 0040112E
00401110	-	8B4D F0	mov	ecx, dword ptr [ebp-10]

堆栈地址=0012FE00, (ASCII "tUiat")

edx=000109AB

转移来自 004010F0, 004010FB

这里是直接字符串比较，第二部分是tUlat，接下来是第三部分：

004015E4	~	75 3F	jnz	short 00401625	
004015E6	.	0FBE85 DDFEF1	movsx	eax, byte ptr [ebp-123]	
004015ED	.	0FBE8D F2FEF1	movsx	ecx, byte ptr [ebp-10E]	
004015F4	.	33C1	xor	eax, ecx	
004015F6	.	83F8 3A	cmp	eax, 3A	
004015F9	~	75 2A	jnz	short 00401625	
004015FB	.	0FBE95 DEFEF1	movsx	edx, byte ptr [ebp-122]	
00401602	.	0FBE85 F1FEF1	movsx	eax, byte ptr [ebp-10F]	
00401609	.	33D0	xor	edx, eax	
0040160B	.	83FA 0B	cmp	edx, 0B	
0040160E	~	75 15	jnz	short 00401625	
00401610	.	0FBE8D DFFEF1	movsx	ecx, byte ptr [ebp-121]	
00401617	.	0FBE95 F0FEF1	movsx	edx, byte ptr [ebp-110]	
0040161E	.	33CA	xor	ecx, edx	
00401620	.	83F9 2D	cmp	ecx, 2D	
00401623	.	74 12	ja	short 00401637	

得到最后的 Flag: XDCTF{Congra\_tUlat\$eyOu}

## Reverse 500

这题首先考虑如何断下来, 点击注册后, 会弹出注册码格式错误的对话框, 于是使用运行+弹框+“ALF+F9”的方法回到用户代码:

00407DB5	>	6A 00	push	0		Style = MB_OK MB_APPLMODAL
00407DB7	.	68 24BD4200	push	0042BD24		错误
00407DBC	.	68 38BD4200	push	0042BD38		长度非法或输入字符无效
00407DC1	.	6A 00	push	0		hOwner = NULL
00407DC3	.	FF15 48814100	call	dword ptr [ &USER32.MessageBoxA		MessageBoxA
00407DC9	.	33C0	xor	eax, eax		

往上找找看到按钮函数的入口:

00407CDB	>	8B35 90814100	mov	esi, dword ptr [ &USER32.GetDlgItemTextA	USER32.GetDlgItemTextA
00407CDF	.	8D4424 78	lea	eax, dword ptr [esp+78]	
00407CE3	.	6A 64	push	64	Count = 64 (100.)
00407CE5	.	50	push	eax	Buffer
00407CE6	.	68 E9030000	push	3E9	ControlID = 3E9 (1001.)
00407CEB	.	57	push	edi	hWnd
00407CEE	.	FFD6	call	esi	GetDlgItemTextA
00407CF0	.	6A 64	push	64	Count = 64 (100.)
00407CF8	.	8D4424 7C	lea	eax, dword ptr [esp+7C]	
00407CF5	.	50	push	eax	Buffer
00407CF5	.	68 66270000	push	2766	ControlID = 2766 (10086.)
00407CFA	.	57	push	edi	hWnd
00407CFB	.	FFD6	call	esi	GetDlgItemTextA
00407CFD	.	85C0	test	eax, eax	
00407CFE	.	0F84 80000000	ja	00407DB5	len(key)=48
00407D05	.	6A 64	push	64	Count = 64 (100.)
00407D07	.	8D4424 7C	lea	eax, dword ptr [esp+7C]	
00407D08	.	50	push	eax	Buffer
00407D0C	.	68 1A270000	push	271A	ControlID = 271A (10010.)
00407D11	.	57	push	edi	hWnd
00407D12	.	FFD6	call	esi	GetDlgItemTextA

看到这里愣了半天, 因为只有两个 Edit 框, 但是 ID 却有两个不是控件的 ID, 而且上来就会挂在这个函数的返回值中……后来手动 fuzz 了一下, 发现输入 47 个 1 和 1 个 0 会成功跳过两个判断……然而, 有次手贱跟进去才发现, 似乎有个 jmp 到了一个奇怪的地方:

004079C0	.	55	push	ebp	
004079C1	.	8BEC	mov	ebp, esp	
004079C3	.	8B45 0C	mov	eax, dword ptr [ebp+C]	
004079C6	.	3D 66270000	cmp	eax, 2766	
004079CB	.	75 1B	jnz	short 004079E8	
004079CD	.	8B4D 10	mov	ecx, dword ptr [ebp+10]	
004079D0	.	8D51 01	lea	edx, dword ptr [ecx+1]	
004079D3	.	8A01	mov	al, byte ptr [ecx]	
004079D5	.	41	inc	ecx	
004079D6	.	84C0	test	al, al	
004079D8	.	75 F9	jnz	short 004079D3	
004079DA	.	2BCA	csh	ecx, edx	

IDA 看看:

```
1 int __stdcall sub_4079C0(int a1, int a2, const char *a3, int a4)
2 {
3     int result; // eax@2
4
5     if ( a2 == 10086 )
6     {
7         result = strlen(a3) == 48;
8     }
9     else if ( a2 == 10010 )
10    {
11        result = sub_408480(a3);
12    }
13    else
14    {
15        result = ((int (__stdcall *) (int, int, const char *, int))lpAddress)(a1, a2, a3, a4);
16    }
17    return result;
18 }
```

原来是这个套路, 其他两个 GetDlgItemTextA 是用来做长度判断和内容检测, 要求长度为 48 字节且为数字或者大写字母……

然后对数据输入下硬件访问断点, 可以看到调用了 sub\_408480 函数, 做了 hex2bin 的转换:

00407045	- F3:	prefix rep:	
00407046	- 0F7F424 50	movq	qword ptr [esp+50], mm0
00407048	- F3:	prefix rep:	
0040704C	- 0F7E40 10	movd	dword ptr [eax+10], mm0
00407050	- 66	db	66
00407051	- 0F	db	0F
00407052	- D6	salc	
00407053	- 44	inc	esp
00407054	- 24 60	and	al, 60
00407055	- E8 15040000	call	00408170
00407058	- 60 00	nush	0

然后进入解密函数：

```

v15 = 0x99864534;
v16 = 0xA5CD481A;
v9.m128i_i8[0] = 0;
__mm_storel_epi64((__m128i *)&v10, 0i64);
__mm_storeu_si128((__m128i *)((char *)&v9 + 1), 0i64);
v11 = 0;
v12 = 0;
v13 = 0;
sub_408010((__int *)&v15, (__int *)&v9);
v0 = 16;

```

这里将 8 字节密钥 push 进了 sub\_408010，IDA 进去看看：

```

sub_408690("解密之前: ");
v4 = 0;
do
{
    sub_408690("%.2X", v11.m128i_i8[v4++]);
} while ( v4 < 24 );
sub_408690("\n");
v5 = a2;
v6 = &v11.m128i_i8[-a2];
v7 = 3;
do
{
    v10 = *(_DWORD *)&v6[v5];
    v20 = *(_DWORD *)&v6[v5 + 4];
    sub_4015C0((int)&v10, (int)v23, (int)&v10, 0);
    v5 += 8;
    *(_DWORD *)&v5 - 8 = *(_DWORD *)&v23;
    *(_DWORD *)&v5 - 4 = *(_DWORD *)&v23[4];
} while (v7--);

```

这里写了是做解密，同时用处理过的密钥 v10 进行解密，每次解密 8 字节，放到 v5 中。用 PEID 插件看下：



唔……结合 64 位明文组和 8 字节密钥，猜想是 DES ECB 模式解密，写了个 python 脚本对比了下解密结果，发现符合解密情况，那就好办了。接下来看看还进行了什么操作：

00408100	> 807C05 C8 08	cmp	byte ptr [ebp+eax-38], 0
00408105	~ 0F85 8A000000	jnz	00408265
00408108	- 40	inc	eax
0040810C	- 83F8 18	cmp	eax, 18
0040810F	- 7C EF	jl	short 004081D0
004081E1	- 6A 10	push	10
004081E3	- E8 57050000	call	0040873F
004081E8	- 8B 02000000	mov	edx, 2
004081ED	- 8D7D C8	lea	edi, dword ptr [ebp-38]
004081F0	- 83C4 04	add	esp, 4
004081F3	- 8BD8	mov	ebx, eax
004081F5	- 8D72 0E	lea	esi, dword ptr [edx+E]
004081F8	- EB 06	jnp	short 00408200
004081FA	~ 8D9B 00000000	lea	ebx, dword ptr [ebx]
00408200	> 8BCA	mov	ecx, edx
00408202	- 81E1 0F000000	and	ecx, 0000000F
00408208	~ 79 05	jns	short 0040820F
0040820A	- 49	dec	ecx
0040820B	- 83C9 F0	or	ecx, FFFFFFF0
0040820E	- 41	inc	ecx
0040820F	> 804417 FF	mov	al, byte ptr [edi+edx-2]

取最后一组解密结果与 0808080808080808 进行比较，所以先将该组加密结果作为注册

码的最后 16 个 16 进制位，即 970BB701CAF237F3。

接下来：

```
do
{
    u5 = u2 % 16;
    *(&u9.m128i_i8[u2++ - 2]) = u9.m128i_i8[u2++ - 2];
    *(&u3->m128i_i32 + u5) = u5;
    --u4;
}
while ( u4 );
__mm_storeu_si128((__m128i *)&v14, __mm_loadu_si128(u3));
Free(u3->m128i_i32);
u7 = 0;
__mm_storeu_si128(&u9, __mm_xor_si128(__mm_loadu_si128((const __m128i *)&v14), (__m128i)xmmword_42BD90));
while ( *(&u9->m128i_i8[u7]) == u9.m128i_i8[u7] )
{
    ++u7;
    if ( u7 >= 16 )
        return 1;
}
```

这个逻辑就比较清晰了，前 16 字节首先进行移位，赋值给 v14，然后与 0x42bd90 处做异或，然后与 0x430bc0 处（机器码 16 进制字节）做比较，相等即通过。

用 python 写个 keygen 脚本：

```
import pyDes
import binascii
|
machine = 'C541D971DAFAAF3567C09B30CFE9BF9A'.decode('hex')

xor_m = ''

for m in machine:
    xor_m += chr(ord(m)^0xe4)

result = ''

s = 2
for i in range(0, 16):
    result += xor_m[(i+s)%16]

result += ('0808080808080808'.decode('hex'))
#print binascii.b2a_hex(result)

key = '344586991A4BCDA5'.decode('hex')
k = pyDes.des(key)
d = k.encrypt(result)

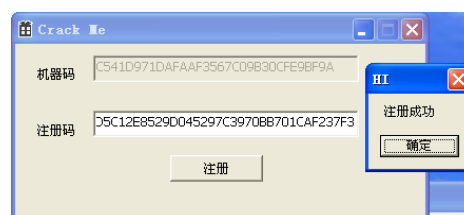
flag = binascii.b2a_hex(d)
Flag = ''

for f in flag:
    if ord(f) in range(ord('a'), ord('z')+1):
        Flag += chr(ord(f)-ord('a')+ord('A'))
    else:
        Flag += f

print Flag
```

算出注册码：282FDA1AD8521D5C12E8529D045297C3970BB701CAF237F3

注册成功，提交之后获得 flag：



## Crypt 200

这题是 AES CBC 字节翻转攻击：

<http://cryptopals.com/sets/2/challenges/16/>

<http://drops.wooyun.org/tips/7828>

基本套路是修改前一个分组对应位置的密文，使用异或之后的结果出现；绕过加密时的限制，但是返回的密文开头并不是第一个字节，所以往后爆破了一下，脚本如下：

```

send_buf = 'mkprof:'
data1 = '@admin=true'

prefix = "comment1=wowsuch%20CBC;userdata="
suffix = ";coment2=%20suchsafe%20very%20encryptwowww"

print len(prefix), len(suffix), len(data1)

sock.send(send_buf + data1 + '\n')
# 接收消息
response_data = sock.recv(1024)
print response_data, len(response_data)
print data1

for i in range(0, 32):
    first = response_data[2*i:2*i+2].decode('hex')

    change_bit = chr(ord(first) ^ ord(';') ^ ord('@'))
    data2 = response_data[0:2*i] + change_bit.encode('hex') + response_data[2*i+

    print data2

    send_buf = 'parse:'
    sock.send(send_buf + data2 + '\n')
    ret = sock.recv(1024)
    #print data2
    print ret

# 接收消息
response_data = sock.recv(1024)
print 'recv:' + response_data

```

最后跑出来 flag:

```

Parsing Profile...You are a normal user.

684299166a05383e6aaa9139f8d8f5848cda560698b1987eb2092534397496b7f49b2991e0d7cfd5
0863843548ef0f9a6924069ce5564a8c832f3f9d36f49990b85a449a7a5234fba0315c69e63323d3
996be3d42536ealcb9c7b4ea7896428d

Parsing Profile...You are a normal user.

684299166a05383e6aaa9139f8d8f5fff7da560698b1987eb2092534397496b7f49b2991e0d7cfd5
0863843548ef0f9a6924069ce5564a8c832f3f9d36f49990b85a449a7a5234fba0315c69e63323d3
996be3d42536ealcb9c7b4ea7896428d

Parsing Profile...Congratulations!
The FLAG is: XDCTF{WelcometToCyberEngineeringSchool2333}
684299166a05383e6aaa9139f8d8f5ff8ca1560698b1987eb2092534397496b7f49b2991e0d7cfd5
0863843548ef0f9a6924069ce5564a8c832f3f9d36f49990b85a449a7a5234fba0315c69e63323d3
996be3d42536ealcb9c7b4ea7896428d

```

## Web1 100

神奇的 MD5。。。刚开始以为真的让碰撞。。。然而因为没猜到 test=是让干嘛，就猜可能是能拿到代码，试了一些可能导致代码泄漏的方式试验，最后发现 test.php~可以看到源码，但是源码是被加密的，百度搜了一个解密的网站 <http://tool.lu/php/> 解密之后看到代码，发现要绕过 \$test=md5(\$test); if(\$test=='0') ，这个可以利用 PHP 的科学计数法绕过

搜到这个帖子 <http://stackoverflow.com/questions/22140204/why-md5240610708-is-equal-to-md5qnkcdzo>

payload:QNKCDZO

## Web1 200

查看网站页面代码，发现/examples/目录

Tomcat 的 examples 页面未删除，导致 session 可以被设置，根据 examples 页面的表单猜出 user=Administrator，设置访问后发现说没有 login，然后设置 login=true



← → ↻ [flagbox-23031374.xdctf.win:1234/examples/servlets/servlet/SessionExample](http://flagbox-23031374.xdctf.win:1234/examples/servlets/servlet/SessionExample)

### Sessions Example

Session ID: 3ED3AEE8016F24337ECD30511F0B403B  
Created: Sat Oct 03 10:59:09 JST 2015  
Last Accessed: Sat Oct 03 11:00:30 JST 2015

The following data is in your session:  
`login = true`  
`user = Administrator`

Name of Session Attribute:   
Value of Session Attribute:

重新访问 <http://flagbox-23031374.xdctf.win:1234/examples/>  
即可拿到 flag

## Web1 400

能输入的就一个登录，其他啥东西都没有。还有个 SleepzzzzZZZ 是什么鬼！

Robots.txt 中发现提示：

Tips:

Brute-force cracking is not usefull!  
thinking again and again! Good luck!

恩，和没说一样。。。。

仔细查看页面源码，发现 freebuf 这个图片源自 picture.php



```
</p><h1> WIF WORK SPACE!
```

访问之，发现图片末尾的提示：

```
<!--Please input the ID as parameter with numeric value-->
```

<http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/Picture.php?ID=1>

id=1 时返回图片，=2 时返回 not found，猜测为 sql 注入

根据提示 双引号

构造 payload:

2" or 1=1%23 返回图片  
2" or 1=2%23 返回 not found  
证实注入。

测试发现，后台暴力过滤了 select union and sleep benchmark ascii substr instr mid left right 等一系列关键字，出现黑名单中的关键字就会直接返回 ID=1 时的图片，不执行查询。

( ∪ ° ∩ ° ) ∪ ∩ ———  
要不要这样。。。。把 select 去掉我就不会玩了好么。。。

使用各种姿势尝试绕过 select 过滤无果后，惊奇地发现原本查询的表中有 username 字段：  
ID=1" %26%26 username is not null%23 返回图片  
同样的姿势发现 password 字段

盲注猜之，  
ID=1" %26%26 username =' admin' %23  
ID=1" %26%26 password =' 5832f4251cb6f43917df' %23

20 位 md5，估计不是登录密码。  
按 dedecms 加密格式，去掉前三位后一位，向土豪乞讨 1 毛，解码 2f4251cb6f43917d  
得到 lu5631209

登录之，获得 flag: XDCTF{e0a345cadaba033073d88d2cc5dce2f7}

## Web2 200

源码泄漏，用 ript-git.pl

拉回一堆.git 下的文件

在 README 中发现提示

All source files are in git tag 1.0

去题目站点下载.git\refs\tags\1.0 丢进对应目录

导出打了 tag 1.0 的文件

```
$ git archive --format=zip --output=1.zip 1.0
```

获得源码，源码中 index.php 包含 flag

```
<?php  
/*
```

Congratulation, this is the [XDSEC-CMS] flag 1

XDCTF-{raGWvWahqZjww4RdHN90}

```
*/
```

```
echo "Hello World";
```

```
?>
```

## Web2 100

提示为逻辑漏洞

重置密码的逻辑位于:

`xdsec_app\front_app\controllers\Auth.php`

```
public function handle_resetpwd()
{
    if(empty($_GET["email"]) || empty($_GET["verify"])) {
        $this->error("Bad request", site_url("auth/forgetpwd"));
    }
    $user = $this->user->get_user(I("get.email"), "email");
    if(I('get.verify') != $user['verify']) {
        $this->error("Your verify code is error", site_url('auth/forgetpwd'));
    }
    if($this->input->method() == "post") {
        $password = I("post.password");
        if(!$this->confirm_password($password)) {
            $this->error("Confirm password error");
        }
        if(!$this->complex_password($password)) {
            $this->error("Password must have at least one alpha and one number");
        }
        if(strlen($password) < 8) {
            $this->error("The Password field must be at least 8 characters in length");
        }
        $this->user->update_userinfo([
            "password" => $password,
            "verify" => null
        ], $user["uid"]);
        $this->success("Password update successful!", site_url("auth/login"));
    } else {
        $url = site_url("auth/resetpwd") . "?email={$user['email']}&verify={$user['verify']}";
        $this->view("resetpwd.html", ["form_url" => $url]);
    }
}
```

重置密码链接由邮箱+随机 verify 组成,但重置成功时会把 verify 重置为 null

题目网站右键源码可以发现作者的邮箱地址:

```
<meta name="author" content="xdsec-cms@xdctf.com"/>
```

于是访问链接:

<http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify=%20>

可重置管理员密码 (貌似管理员的 verify 一直是 null 不会改变)

登录后可查看 flag 文件:

Congratulation, this is the [XDSEC-CMS] flag 2

XDCTF-{i32mX4WK1gwEE9S90xd2}

hint:

admin url is /th3r315admln.php