

# 网鼎杯青龙组逆向writeup

原创

[wumingpeng](#) 于 2020-05-18 17:18:50 发布 312 收藏  
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/wumingpeng/article/details/106196876>  
版权

bang

1. 配置frida环境：安装python3， pip install frida ， pip install fridatools。

在<https://github.com/frida/frida/releases>下载对应的frida-server

[frida-server-12.8.20-android-arm.xz](#)

[frida-server-12.8.20-android-arm64.xz](#)

[frida-server-12.8.20-android-x86.xz](#)

[frida-server-12.8.20-android-x86\\_64.xz](#)

用下面命令把frida-server push到手机或模拟器中

```
adb push frida-server /data/local/tmp
```

```
adb chmod 777 /data/local/tmp/frida-server
```

执行frida-server

再开一个命令行窗口，执行adb forward tcp:27042 tcp:27042

```
C:\Users\wmp10>adb shell
root@shamu:/ # chmod 777 /data/local/tmp/frida-server
chmod 777 /data/local/tmp/frida-server
root@shamu:/ # /data/local/tmp/frida
/data/local/tmp/frida
/system/bin/sh: /data/local/tmp/frida: not found
127|root@shamu:/ # /data/local/tmp/frida-server
/data/local/tmp/frida-server
WARNING: linker: /data/local/tmp/frida-server: unused DT entry: type 0x6ffffef5 arg 0x1c24
```

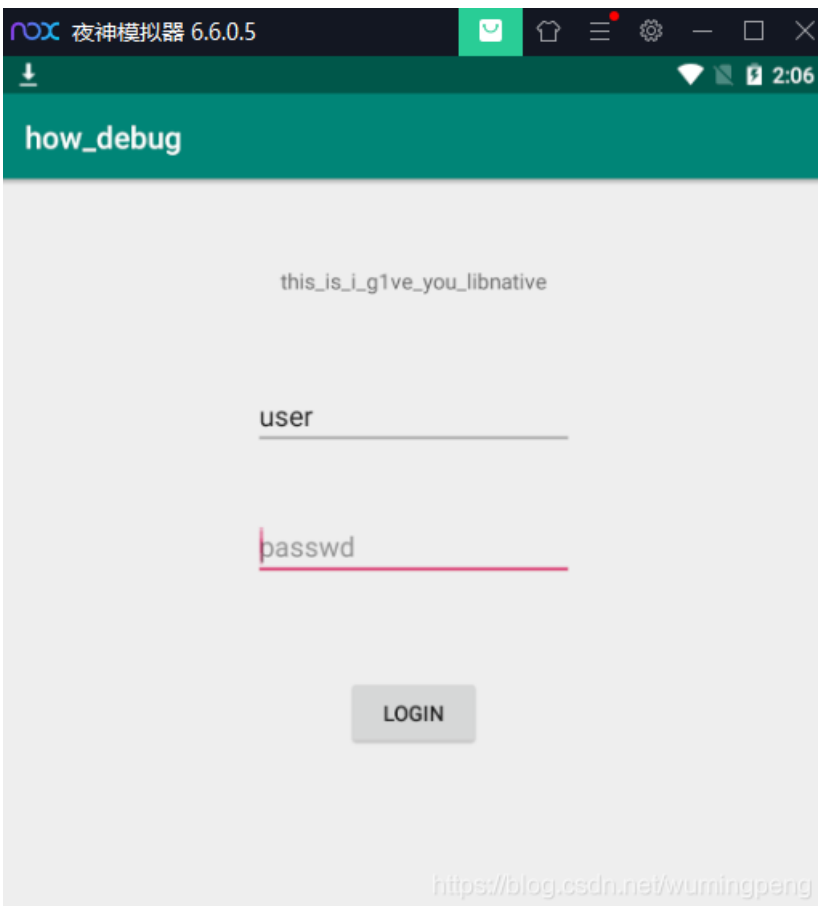
```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.18362.778]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\wmp10>adb forward tcp:27042 tcp:27042

C:\Users\wmp10>
```

<https://blog.csdn.net/wumingpeng>

然后再模拟器里面启动apk

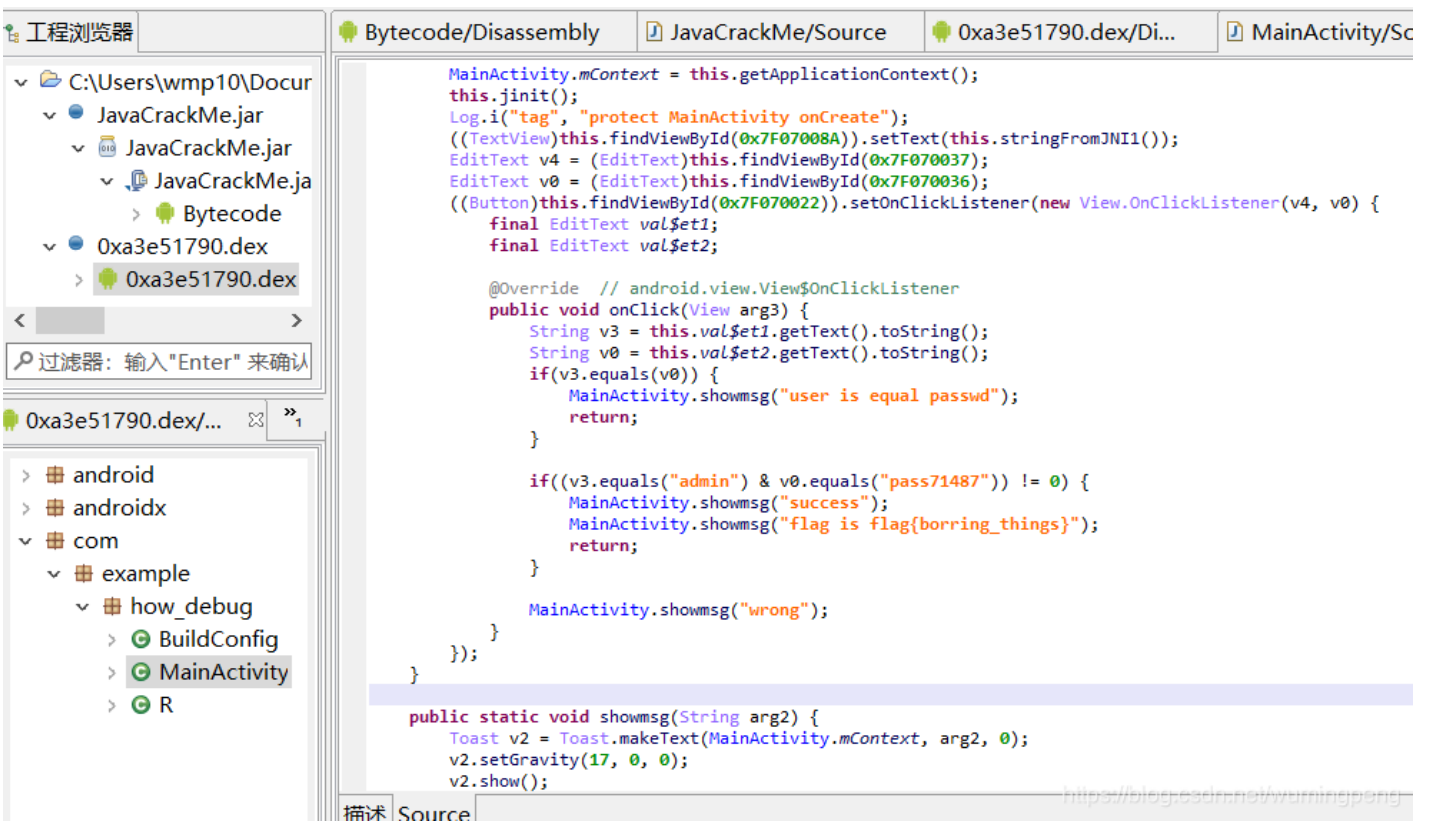


然后运行脱壳工具

```
C:\Windows\system32\cmd.exe
D:\Tool\FRIDA-DEXDump-master>python main.py
05-13/14:06:49 INFO [DEXDump]: found target [3050] com.example.how_debug
[DEXDump]: DexSize=0x1d60b8, SavePath=D:\Tool\FRIDA-DEXDump-master\com.example.how_debug\0xa3e51790.dex
[DEXDump]: DexSize=0x446c, SavePath=D:\Tool\FRIDA-DEXDump-master\com.example.how_debug\0xb329f000.dex
[DEXDump]: DexSize=0x446c, SavePath=D:\Tool\FRIDA-DEXDump-master\com.example.how_debug\0xbfe4519c.dex
D:\Tool\FRIDA-DEXDump-master>
```

<https://blog.csdn.net/wumingpeng>

1. 用jeb反编译脱壳的dex,就可以看到flag



```
MainActivity.mContext = this.getApplicationContext();
this.jinit();
Log.i("tag", "protect MainActivity onCreate");
((TextView)this.findViewById(0x7F07008A)).setText(this.stringFromJNI1());
EditText v4 = (EditText)this.findViewById(0x7F070037);
EditText v0 = (EditText)this.findViewById(0x7F070036);
((Button)this.findViewById(0x7F070022)).setOnClickListener(new View.OnClickListener(v4, v0) {
    final EditText val$set1;
    final EditText val$set2;

    @Override // android.view.View$OnClickListener
    public void onClick(View arg3) {
        String v3 = this.val$set1.getText().toString();
        String v0 = this.val$set2.getText().toString();
        if(v3.equals(v0)) {
            MainActivity.showmsg("user is equal passwd");
            return;
        }

        if((v3.equals("admin") & v0.equals("pass71487")) != 0) {
            MainActivity.showmsg("success");
            MainActivity.showmsg("flag is flag{borring_things}");
            return;
        }

        MainActivity.showmsg("wrong");
    }
});
}

public static void showmsg(String arg2) {
    Toast v2 = Toast.makeText(MainActivity.mContext, arg2, 0);
    v2.setGravity(17, 0, 0);
    v2.show();
}
```

<https://blog.csdn.net/wumingpeng>

Jocker

1.首先在程序的开头就把代码段的一段空间修改为可读可写的属性。代码段本身属性是可读可执行，不可写的。

```

lea    ecx, [esp+4]
and    esp, 0FFFFFF0h
push   dword ptr [ecx-4]
push   ebp
mov    ebp, esp
push   ecx
sub    esp, 0A4h
call   __main
mov    dword ptr [esp], offset aPleaseInputYou ; "please input you flag:"
call   _puts
lea    eax, [ebp+f101dProtect]
mov    [esp+0Ch], eax ; lpf101dProtect
mov    dword ptr [esp+8], 4 ; flNewProtect
mov    dword ptr [esp+4], 0C8h ; dwSize
mov    dword ptr [esp], offset _27encryptPc ; lpAddress
mov    eax, ds:__imp__VirtualProtect@16 ; VirtualProtect(x,x,x,x)
call   eax ; VirtualProtect(x,x,x,x) ; VirtualProtect(x,x,x,x)
sub    esp, 10h
test   eax, eax
setz   al
test   al, al
jz     short loc_401788

```

<https://blog.csdn.net/wumingpeng>

2.比较输入的flag字符串长度是否为24

```

loc_401788:
lea    eax, [ebp+inputFlag]
mov    [esp+4], eax
mov    dword ptr [esp], offset a40s ; "%40s"
call   _scanf
lea    eax, [ebp+inputFlag]
mov    [esp], eax ; char *
call   _strlen
mov    [ebp+inputFlagLen], eax
cmp    [ebp+inputFlagLen], 18h
jz     short loc_4017CD

```

3.有两段伪加解密和比较伪flag函数

```
char flag[]={f',k',c',d',0x7f,'a','g',d',0x3b,'v','k','a',0x7b,0x26,0x3b,0x50,0x63,0x5f,0x4d,0x5a,0x71,
```

```
0xc,0x37,0x66};
```

```
for(int i=0;i<=23;i++)
```

```
{if(i&1)
```

```
{flag[i]=flag[i]+i; }
```

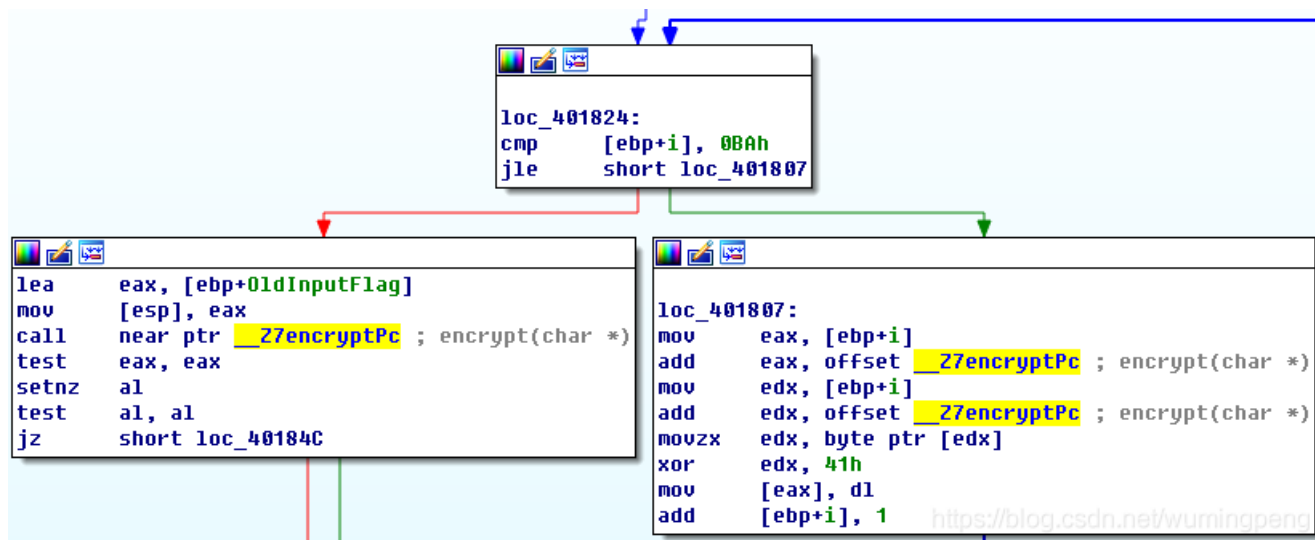
```
else
```

```
{flag[i]=flag[i]^i;
```

```
}
```

```
}//flag{fak3_alw35_sp_me!!}
```

4.解密真正的flag算法和比较代码，也就是1里面修改了内存属性的那一段地址代码。



下图是解密前的代码

```
;_DWORD __cdecl encrypt(char *)
public __Z7encryptPc
__Z7encryptPc proc far
adc    al, 0C8h
movsb
push  ss
pop   ss
adc    al, dl
lodsd
cmp    eax, 40A10486h
inc    ecx
inc    ecx
inc    ecx
int    3 ; Trap to Debugger
add    al, 0D5h
cli
add    [ecx+1], esi
inc    ecx
sti
push  edx
inc    ecx
inc    ecx
inc    ecx
enter  0FFFFC886h, 9Fh
enter  0FFFFB290h, 0E4h
xchg  al, ds:41414141h
stosb
or    dl, cl
adc    al, 0A5h
retf  4904h
```

5写脚本解密代码

stadd=0x401500

stsize=186

i=0

while(i<=186):

```
ch=ord(ida_bytes.get_bytes(stadd+i,1))^0x41
```

```
ida_bytes.patch_byte(stadd+i,ch)
```

```
i+=1
```

```
print("done")
```

下图是解密后真正计算和比较flag的代码

```
int __cdecl encrypt(char *a1)
{
    int v2[19]; // [esp+1Ch] [ebp-6Ch]
    int v3; // [esp+68h] [ebp-20h]
    int i; // [esp+6Ch] [ebp-1Ch]

    v3 = 1;
    memcpy(v2, &unk_403040, sizeof(v2));
    for ( i = 0; i <= 18; ++i )
    {
        if ( (char)(a1[i] ^ aHahahaDoYouF[i]) != v2[i] )
        {
            puts("wrong ~");
            v3 = 0;
            exit(0);
        }
    }
    if ( v3 == 1 )
        puts("come here");
    return v3;
}
```

<https://blog.csdn.net/wumingpeng>

00401500	55	push	ebp	
00401501	89E5	mov	ebp, esp	
00401503	57	push	edi	
00401504	56	push	esi	
00401505	53	push	ebx	
00401506	83EC 7C	sub	esp, 7C	
00401509	C745 E0 010000	mov	dword ptr [ebp-20], 1	
00401510	8D45 94	lea	eax, [ebp-6C]	
00401513	BB 40304000	mov	ebx, 00403040	
00401518	BA 13000000	mov	edx, 13	
0040151D	89C7	mov	edi, eax	
0040151F	89DE	mov	esi, ebx	
00401521	89D1	mov	ecx, edx	
00401523	F3:A5	rep	movs dword ptr es:[edi], dword ptr [esi]	
00401525	C745 E4 000000	mov	dword ptr [ebp-1C], 0	
0040152C	EB 49	jmp	short 00401577	
0040152E	8B55 E4	mov	edx, [ebp-1C]	
00401531	8B45 08	mov	eax, [ebp+8]	
00401534	01D0	add	eax, edx	
00401536	0FB610	movzx	edx, byte ptr [eax]	
00401539	8B45 E4	mov	eax, [ebp-1C]	
0040153C	05 12404000	add	eax, 00404012	ASCII "hahahaha_do_you_find_me?"
00401541	0FB600	movzx	eax, byte ptr [eax]	
00401544	31D0	xor	eax, edx	
00401546	0FBED0	movsx	edx, al	
00401549	8B45 E4	mov	eax, [ebp-1C]	
0040154C	8B4485 94	mov	eax, [ebp+eax*4-6C]	
00401550	39C2	cmp	edx, eax	
00401552	74 1F	je	short 00401573	
00401554	C70424 00404000	mov	dword ptr [esp], 00404000	ASCII "wrong ~"
0040155B	E8 E0130000	call	<jmp.&msvcrt.puts>	
00401560	C745 E0 000000	mov	dword ptr [ebp-20], 0	
00401567	C70424 00000000	mov	dword ptr [esp], 0	
0040156E	E8 AD130000	call	<jmp.&msvcrt.exit>	
00401573	8345 E4 01	add	dword ptr [ebp-1C], 1	
00401577	837D E4 12	cmp	dword ptr [ebp-1C], 12	
0040157B	7E B1	jle	short 0040152E	
0040157D	837D E0 01	cmp	dword ptr [ebp-20], 1	
00401581	75 0C	jnz	short 0040158F	
00401583	C70424 00404000	mov	dword ptr [esp], 00404000	ASCII "come here"
0040158A	E8 B1130000	call	<jmp.&msvcrt.puts>	<a href="https://blog.csdn.net/wumingpeng">https://blog.csdn.net/wumingpeng</a>

## 5算法

取输入的flag的19个字符与hahahaha\_do\_you\_find\_me?的前19个字符做异或运算

比较值是否为

```
{0xe,0xd,0x9,0x6,0x13,0x5,0x58,0x56,0x3e,0x6,0xc,0x3c,0x1f,0x57,0x14,0x6b,0x57,0x59,0xd}
```

如果相等就是正确的flag。

```
char key[]={"hahahaha_do_you_find_me?"};
```

```
char enflag[]={0xe,0xd,0x9,0x6,0x13,0x5,0x58,0x56,0x3e,0x6,0xc,0x3c,0x1f,0x57,0x14,0x6b,0x57,0x59,0xd};
```

```
char a;
```

```
for(int i=0;i<20;i++)
```

```
{
```

```
    a = key[i]^enflag[i];
```

```
    printf("%c\n",a);
```

```
}
```

```
a = [0x25,0x74,0x70,0x26,0x3a]
```

```
flag = 'flag{d07abccf8a410c'
```

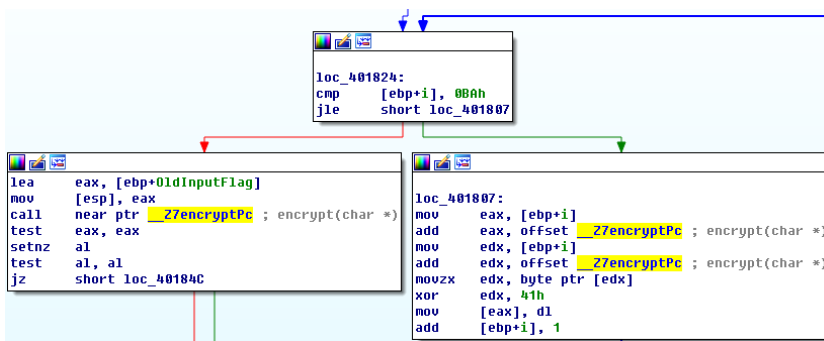
```
for i in range(len(a)):
```

```
    flag += chr(a[i]^71)
```

```
print(flag)
```

## Signal

1.是一个简单的虚拟机运行的题目



2.通过虚拟机的流程控制去执行相应的操作

59	17	40	00	C5	15	40	00	E7	15	40	00	19	16	40	00
4D	16	40	00	7D	16	40	00	B4	16	40	00	BD	16	40	00
3C	17	40	00	59	17	40	00	AE	15	40	00	08	17	40	00
22	17	40	00	67	6F	6F	64	2C	54	68	65	20	61	6E	73

3.流程控制执行的代码数组，下面的代码地址对应相应的case

59	17	40	00	C5	15	40	00	E7	15	40	00	19	16	40	00
4D	16	40	00	7D	16	40	00	B4	16	40	00	BD	16	40	00
3C	17	40	00	59	17	40	00	AE	15	40	00	08	17	40	00
22	17	40	00	67	6F	6F	64	2C	54	68	65	20	61	6E	73

4.下面是程序对输入的flag做的运算处理，计算每一个字符与 {0x22,0x3f,0x34,0x32,0x72,0x33,0x18,0xfa7,0x31,0xf1,0x28,0xf84,0xfc1,0x1e,0x7a}去做对比

1	$10h * input[0] - 5 = 22h$
2	$(20h * input[1]) * 3 = 3Fh$
3	$input[2] - 2 - 1 = 34h$
4	$(input[3] + 1)^4 = 32h$
5	$input[4] * 3 - 21h = 72h$
6	$input[5] - 1 - 1 = 33h$
7	$9 * input[6] - 20 = 18$
8	$(51h + input[7])^{24h} = FA7$
9	$input[8] + 1 - 1 = 31h$
10	$2 * input[9] + 25h = F1$
11	$(36h + input[10])^{41h} = 28h$
12	$(20h + input[11]) * 1 = F84h$
13	$3 * input[12] + 25h = C1h$
14	$9 * input[13] - 20h = 1Eh$
15	$41h + input[14] + 1 = 7Ah$

5.还原flag

```
int main(int argc, char* argv[])
{
    printf("%c", (0x10^(0x22+5)));
    printf("%c", (0x20^(0x3f/3)));
    printf("%c", (0x34+3));
    printf("%c", ((0x32^4)-1));
    printf("%c", ((0x72+0x21)/3));
    printf("%c", (0x33+2));
    printf("%c", (0x9^(0x18+0x20)));
}
```



```
printf("%c",((0xfa7^0x24)-0x51));
```

```
printf("%c",0x31);
```

```
printf("%c",((0xf1-0x25)/2));
```

```
printf("%c",((0x28^0x41)-0x36));
```

```
printf("%c",0xf84-0x20);
```

```
printf("%c",((0xc1-0x25)/3));
```

```
printf("%c",((0x1e+0x20)^0x9));
```

```
printf("%c\n",0x7a-0x42);
```

```
return 0;
```

```
}
```