西普部分CTF题目（逆向）

原创

gwenchill 于 2015-07-12 21:36:06 发布 16018 收藏 2

分类专栏： CTF学习

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/gwenchill/article/details/46853561

版权

CTF学习 专栏收录该内容

9 篇文章 1 订阅

订阅专栏

1、阿拉丁神灯http://ctf1.simplexue.com/crack/1/

这个比较简单，用ida加载后，函数列表里有Button1_click，点击进去后发现

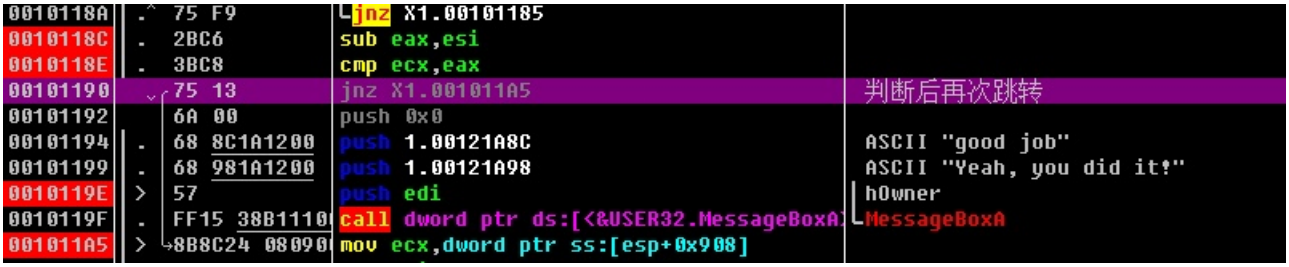
```
.method private instance void Button1_Click(object sender, class [mscorlib]System.EventArgs e)
{
  .maxstack 3
  .locals init (string V0)
darg.0
allvirt instance class [System.Windows.Forms]System.Windows.Forms.TextBox WindowsApplication1.Form1::get_TextBox1()
allvirt instance string [System.Windows.Forms]System.Windows.Forms.TextBox::get_Text()
all     string [Microsoft.VisualBasic]Microsoft.VisualBasic.Strings::Trim(string)
tloc.0
dloc.0
dstr    "zhimakaimen@2011"
dc.i4.0
all     int32 [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Operators::CompareString(string, string, bool)
dc.i4.0
```

这里有个比较函数，应该是输入的字符串和zhimakaimen@2011进行比较，在页面输入该密码，即可获得key

小明向灯神许愿道~ 灯神啊~ 给我过关的Key吧~ 灯神说道\KEY:UnPack&Crack2011!!

2、你知道注册码吗 http://ctf8.simplexue.com/crackme2/

属于crakeme，用吾爱破解打开后，调试运行，输入用户名处输入4个字符，点击注册，可弹出提示。此时调试暂停，选择查看->调用堆栈，找到用户程序段的堆栈，点击进去后，查看程序，发现有一个比较，若不相等则跳转。



在跳转位置101190位置F2设置断点，并观察ecx和eax的值

调试观察发现计算方法是用户名每个字符-8+位置顺序号

python代码为：

username='syclover'

index=0

for c in username:

print chr(ord(c)-8+index)

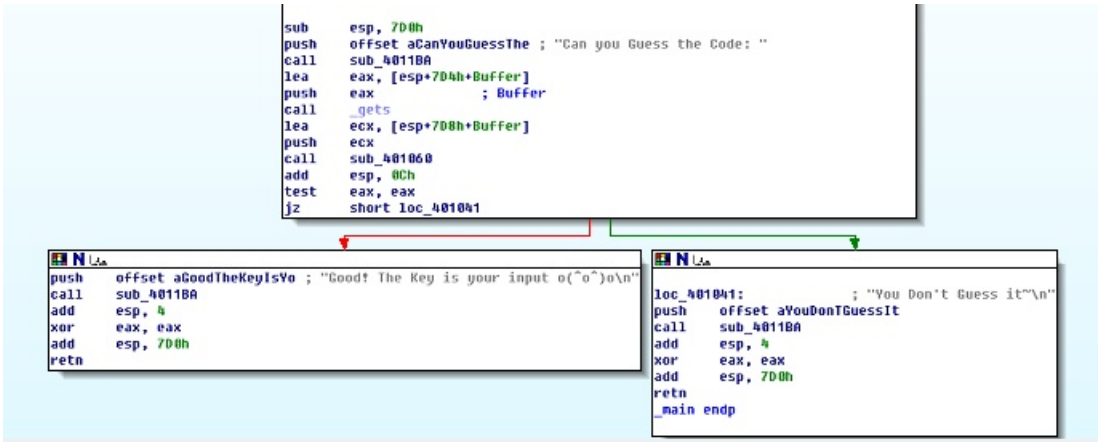
index+=1

输出key： kr]gkscq

3、证明自己 http://ctf8.simplexue.com/crackme/

需要逆向的程序为命令行程序，无界面，结果相对简单。

用IDA打开，根据结构很快可看到整体结构,明显是输入个字符串，判断后返回正确还是错误。判断程序为401060地址开始程序。



进入401060，按F5转换为C程序，可看到原始字符进行异或0x20运算，系统匹配的字符串每个字符-5，再进行比较。进行程序调试，看到某个地方直接跳转结束，判断条件是比较。

在比较句上设置断点，调试后发现edx总为14，若exc不为14直接返回失败，所以输入的字符串长度应该是14个字符。下面loc_4010F7有对系统字符串做-5的运算，在保存运算后字符指令后设置断点，观察每次循环保存的数据，共循环14次。

```
lea     edi, [esp+1Ch+var_10]
add     al, 0FBh        减5
or      ecx, 0FFFFFFFFh
mov     byte ptr [esp+edx+1Ch+var_10], a   保存
xor     eax, eax
inc     edx
repne scasb
not     ecx
dec     ecx
cmp     edx, ecx
jb      short loc_4010E7
```

14次循环得到的数据[0x63,0x52,0x14,0x43,0x4B,0x69,0x53,0x73,0x4F,0x65,0x14,0x53,0x59,0x1]

用python程序简单运算下即可得到

a=[0x63,0x52,0x14,0x43,0x4B,0x69,0x53,0x73,0x4F,0x65,0x14,0x53,0x59,0x1]

for i in a:

print chr(i^0x20),

结果是Cr4ckIsSoE4sy!

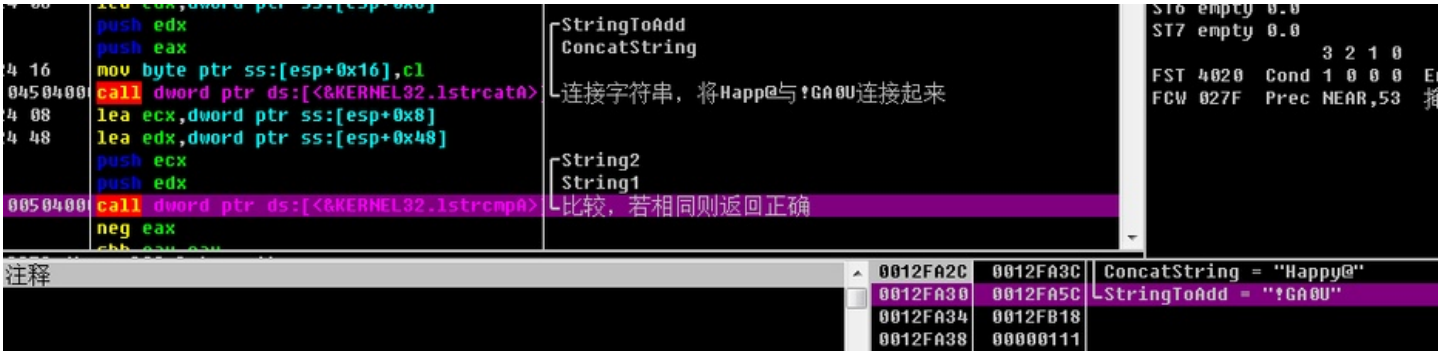
4、该题不简单 http://ctf1.simplexue.com/crack/3/

其实也挺简单的

使用IDA打开程序，找到处理函数，F5转换为C代码，很容易识别，算法是在输入的用户名上进行运算再加上”Happy@”即可，运算方法是：(i+i*char*char)%0x42+33

结果是Happy@!GA0U。

用吾爱破解也可以很容易判别，如下图



在比较处设置断点，右下角的数据段直接显示出计算的字符串和拼接的字符串，字符串连接后即为key。

5、此处无声，http://ctf1.simplexue.com/crack/5/

比较复杂，程序还加了壳，脱壳后调试发现是个算法加密，应该是用MD5,RC6加密，看了半天，算法太复杂，没研究出来，算了，回头逆向理解更深了再说。期待有高手分享经验。

计算方法是RC6Decry(MD5(‘nsfocus’))=Key,RC6的密钥是

6、Flag:http://ctf5.simplexue.com/qwctf/flag-checker.html

该题目给了个javascript，很长的计算公式，字符串长度必须为47位，且满足一面47个判断，其实判断都不复杂，大致推出a[0],再推a[1]，一直到a[46]，求得flag为flag{wh47_my5ter10us-do3s,the+phe45ant/c0nta1n}

使用脚本计算如下：

```
<script language="javascript">
ss="a[11]-a[5]%a[1]*a[12]%a[14]-a省略后面一大串";

xx=ss.split('&&');

function search(num)
{
    index=-1;
    for(i=0;i<xx.length;i++)
    {
        dd=xx[i];
        offset=0;
        count=0
       do{
       offset=dd.indexOf('[',offset);
        if(offset != -1)
        {
        count++;
        offset += 2;
        }
        }while(offset != -1)
        if(count==num)
        {
          index=i;
         break;
        }

    }
    return index;
}

var a = new Array();
a[0]=0;
for(k=0;k<47;k++)
{
    index=search(k+1);
    dd=xx[index];
    for(j=0;j<256;j++)
    {
      a[k]=j;
      if(eval(dd))
         break;
    }
}

jjj='';

for(f=0;f<a.length;f++)
{
   jjj+=String.fromCharCode(a[f]);
}
document.write(jjj);
</script>
```

7、john the packer

题目地址：http://ctf5.simplexue.com/re/topack.html

题目来自意大利的PoliCTF，是linux程序脱壳逆向的，挺新颖。报着学习的态度，理解ELF脱壳逆向。

首先看看是什么文件：

```
xx@kali:~/Desktop$ file topack
topack: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs)
```

是个32位的ELF执行程序，用IDA打开，入口点下翻翻发现：

```
.text:08048633                push    eax
.text:08048634                mov     edx, [edx]
.text:08048636
.text:08048636 loc_8048636:     ; CODE XREF:sub_80485E0+5Cj
.text:08048636                xor     [eax], edx
.text:08048638                add     eax, 4
.text:0804863B                dec     ecx
.text:0804863C                jnz     short loc_8048636
.text:0804863E                pop     eax
.text:0804863F                call    eax
.text:08048641                sub     esp, 8
.text:08048644                push    [ebp+arg_4]
.text:08048647                push    [ebp+arg_0]
.text:0804864A                call    sub_804859B
```

程序有调用call eax的操作，脱壳就是要抓取eax对应的内存。再看调用结束后的函数sub_804859B,代码有：

```
.text:080485D3 loc_80485D3:       ; CODE XREF: sub_804859B+3Ej
.text:080485D3                xor     [ebx], edx
.text:080485D5                add     ebx, 4
.text:080485D8                dec     ecx
.text:080485D9                jnz     short loc_80485D3
```

算了，没耐心写了，参考https://github.com/dqi/ctf_writeup/tree/master/2015/polictf/reversing/john%20the%20packer吧

8、Keylead(ASIS 2015)

file查看keylead文件是个7z文件，解压指令：

```
unxz -d -f keylead -c > keylead1
```

file keylead1发现是个64位ELF文件，IDA64打开，发现sub_400E6E是main函数，查看c代码可看到就是随机数去判断，需要强制更改跳转，一直跳到flag产生处。还可以在函数入口地直接修改地址跳转到产生flag的函数。在函数入口附近找到：

04005DD mov rdi, offset sub_400E6E //跳到主函数

将对应地址修改为flag函数sub_4006B6，用winhex修改保存后为

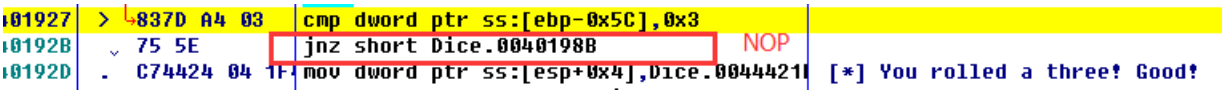
04005DD mov rdi, offset sub_4006B6

在linux中直接直接执行得到flag。

9、bin100(ebCTF2013)

其实同上面说的第8题，程序随机产生一系列随机数，满足3,1,3,3,7的顺序，因此修改跳转即可。

找到winmain函数，用OD调试，强制修改每次判断的eax值为3,1,3,3,7，但输出的flag为乱码。后查询发现时间等待作为参数产生flag，因此需要程序自动完成，不要进行调试。因此nop掉所有的je，类似这样，一共5处。



产生flag的函数前有一个eax与-1比较的判断，把下面的跳转指令nop掉。共6处修改。

用od的菜单–>二进制–>nop填充，应用全部修改，保存到新文件，执行即可得到flag