XCTF-攻防世界CTF平台-Reverse逆向类——1、Mysterious

原创

大灬白 于 2021-08-20 15:48:01 发布 121 收藏 3

分类专栏： #VC++逆向 XCTF-攻防世界-Reverse逆向类题目 文章标签： 字符串 指针 VC++ 缓冲区溢出 CTF

本文链接：https://blog.csdn.net/Onlyone_1314/article/details/119822947

VC++逆向 同时被 2 个专栏收录

2 篇文章 0 订阅

订阅专栏

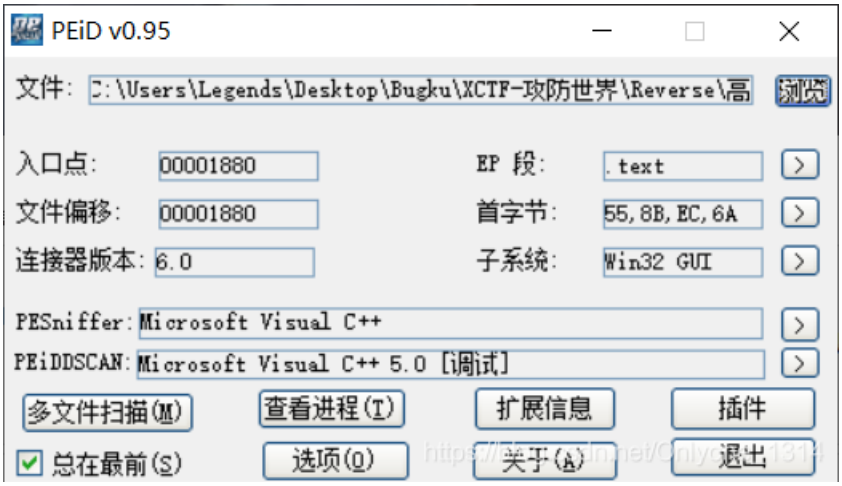
XCTF-攻防世界-Reverse逆向类题目

11 篇文章 1 订阅

订阅专栏

目录标题

先用PEiD查壳：

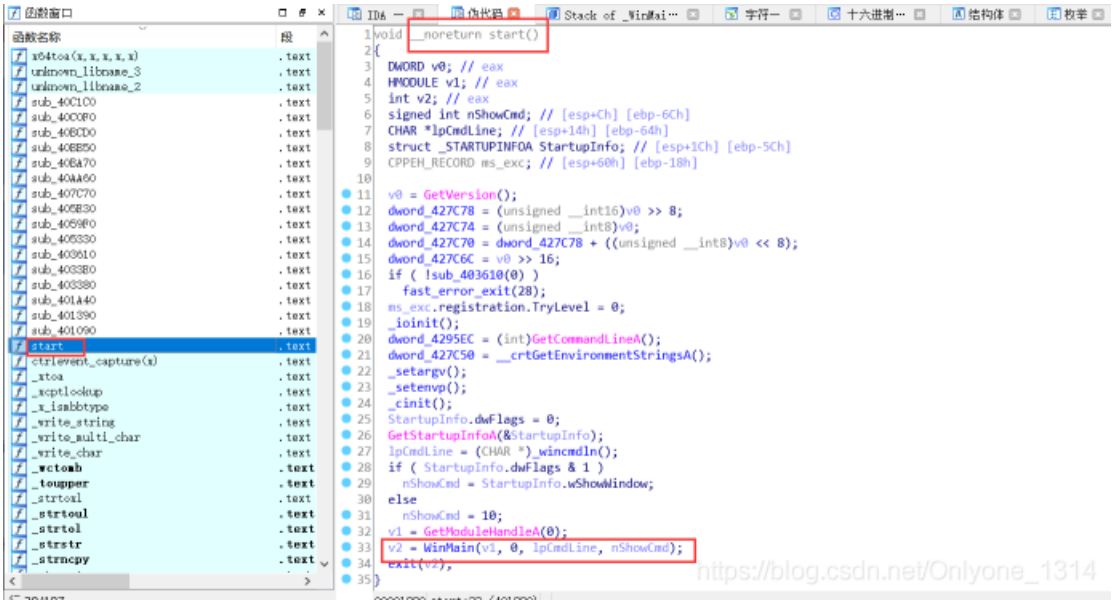


发现没有壳，程序是VC++编写的Win32 GUI程序。
运行程序：

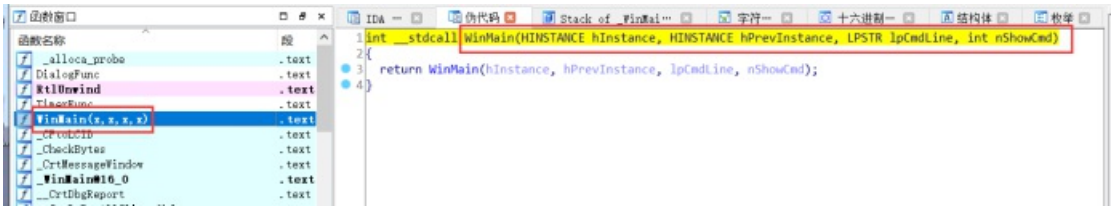
是一个让我们输入密码的程序，输入123点击“Crack”按钮没有反应。

方法1、直接拼凑得到flag

用IDA反编译程序：



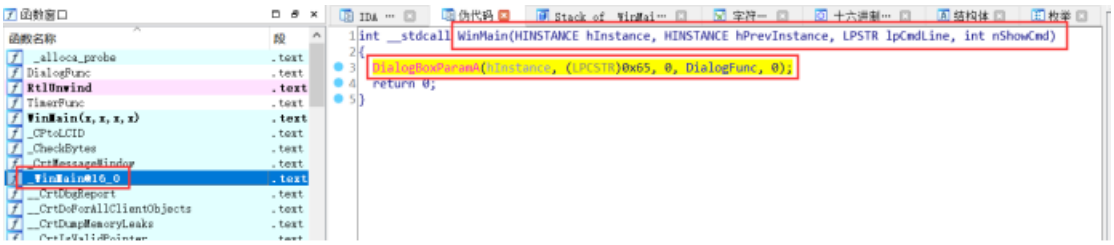
VC++编写的Win32 GUI程序最开始是从start()函数开始，调用：
1、注册异常处理函数
2、调用GetVersion 获取版本信息
3、调用函数 __heap_init初始化堆栈
4、调用 __ioinit函数初始化啊IO环境，这个函数主要在初始化控制台信息，在未调用这个函数之前是不能进行printf的
5、调用 GetCommandLineA函数获取命令行参数
6、调用 GetEnvironmentStringsA 函数获取环境变量
7、调用main函数
再调用表面上的主函数WinMian函数：



因为程序是VC++编写的Win32 GUI程序，与控制台应用程序不同，Win32 GUI程序的入口是WinMian函数（及windows入口函数），而普通的控制台应用程序的入口是main。
WinMian函数原型：

```
int _stdcall WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow
)
```

hinstance：应用程序当前实例的句柄；
hPrevInstance：应用程序的先前实例的句柄。对于同一个程序打开两次，出现两个窗口第一次打开的窗口就是先前实例的窗口。对于一个32位程序，该参数总为NULL；
lpCmdLine：指向应用程序命令行的字符串的指针；
nCmdShow：指明窗口如何显示；
点击再进到下一个程序：



DialogBoxParamA函数根据对话框模板资源创建一个模态的对话框。
这个对话框就是程序出现的主要的对话框：

在显示对话框之前，DialogBoxParamA函数把一个应用程序定义的值作为WM_INITDIALOG消息的lParam参数传到对话框过程,应用程序可用此值来初始化对话框。A代表使用Ansi字符，和W代表使用UNICODE字符对应。

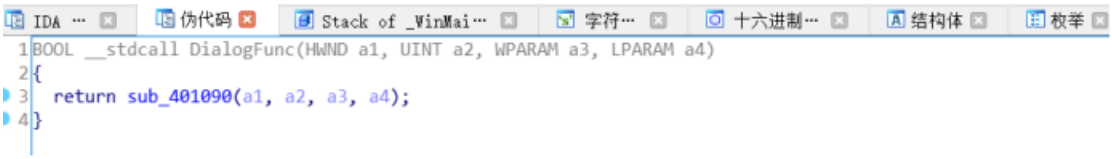
```
DialogBoxParamA(hInstance, (LPCSTR)0x65, 0, DialogFunc, 0);
```

DialogBoxParamA函数原型：

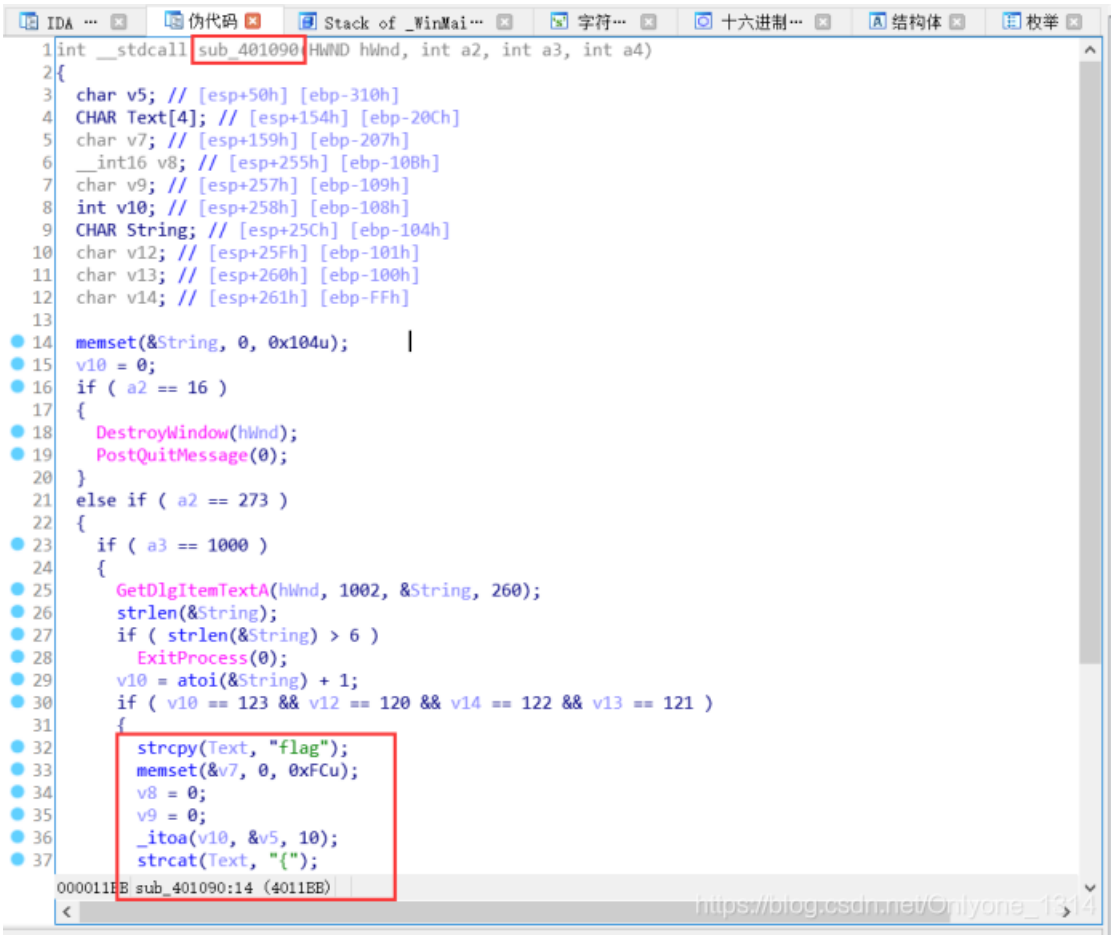
```
int DialogBoxParamA (
HINSTANCE hInstance,
LPCTSTR IpTemplateName,
HWND hWndParent,
DLGPROC IPDialogFunc,
LPARAM dwlnitParam
) ;
```

hInstance：标识应用程序当前实例的句柄；
lpTemplateName：标识对话框模板。LPCSTR是Win32和VC++所使用的一种字符串数据类型。LPCSTR被定义成是一个指向以'\0'结尾的常量字符的指针。指定资源标识符为(LPCSTR)0x65。
hWndParent：指定拥有对话框的窗口。父窗口的句柄，它的子窗口将被搜索。这里hwndParent为0(NULL)，则该函数将桌面窗口作为父窗口。该功能在作为桌面子窗口的窗口之间进行搜索。
lpDialogFunc：指向对话框过程的指针。它的值是DialogFunc(0040100A)函数指针,即调用DialogFunc()函数来处理对话框的消息。
dwlnitParam：指定传递到对话框过程中的 WM_INITDIALOG 消息 lParam 参数的值。这里是初始化参数，这里缺省设置为 0 ；

点击再进到对话框的消息处理函数DialogFunc()：



再进入sub_401090()函数：



这个就是主要的逻辑判断函数。

sub_401090()函数源代码：

```
int __stdcall sub_401090(HWND hWnd, int a2, int a3, int a4)
{
  char v5; // [esp+50h] [ebp-310h]
  CHAR Text[4]; // [esp+154h] [ebp-20Ch]
  char v7; // [esp+159h] [ebp-207h]
  __int16 v8; // [esp+255h] [ebp-10Bh]
  char v9; // [esp+257h] [ebp-109h]
  int v10; // [esp+258h] [ebp-108h]
  CHAR String; // [esp+25Ch] [ebp-104h]
  char v12; // [esp+25Fh] [ebp-101h]
  char v13; // [esp+260h] [ebp-100h]
  char v14; // [esp+261h] [ebp-FFh]

  memset(&String, 0, 0x104u);
  v10 = 0;
  if ( a2 == 16 )
  {
    DestroyWindow(hWnd);
    PostQuitMessage(0);
  }
  else if ( a2 == 273 )
  {
    if ( a3 == 1000 )
    {
      GetDlgItemTextA(hWnd, 1002, &String, 260);
      strlen(&String);
      if ( strlen(&String) > 6 )
        ExitProcess(0);
      v10 = atoi(&String) + 1;
      if ( v10 == 123 && v12 == 120 && v14 == 122 && v13 == 121 )
      {
        strcpy(Text, "flag");
        memset(&v7, 0, 0xFCu);
        v8 = 0;
        v9 = 0;
        _itoa(v10, &v5, 10);
        strcat(Text, "{");
        strcat(Text, &v5);
        strcat(Text, "_");
        strcat(Text, "Buff3r_0v3rf|0w");
        strcat(Text, "}");
        MessageBoxA(0, Text, "well done", 0);
      }
      SetTimer(hWnd, 1u, 0x3E8u, TimerFunc);
    }
    if ( a3 == 1001 )
      KillTimer(hWnd, 1u);
  }
  return 0;
}
```

其中

```
if ( v10 == 123 && v12 == 120 && v14 == 122 && v13 == 121 )
{
  strcpy(Text, "flag");
  memset(&v7, 0, 0xFCu);
  v8 = 0;
  v9 = 0;
  _itoa(v10, &v5, 10);
  strcat(Text, "{");
  strcat(Text, &v5);
  strcat(Text, "_");
  strcat(Text, "Buff3r_0v3rf|0w");
  strcat(Text, "}");
  MessageBoxA(0, Text, "well done", 0);
}
```


这部分代码负责弹出了新的对话框显示flag{&v5_Buff3r_0v3rf|0w}，&v5的值由`_itoa(v10, &v5, 10);`得到，且v10的值等于123

_itoa()函数的作用是取整数输入值，并将其转换为相应进制数字的字符串。v10是int型整数123，所以&v5的值就是字符串“123”，

所以flag就是`flag{123_Buff3r_0v3rf|0w}`。

方法2、逆向计算输入的字符串得到flag

但是我们并不会满足于直接拿到flag这种简单的手段的，接下来尝试逆向计算输入的字符串来得到flag。

带注释的sub_401090()函数源代码：

```
int __stdcall sub_401090(HWND hWnd, int a2, int a3, int a4)
{
  char v5; // [esp+50h] [ebp-310h]
  CHAR Text[4]; // [esp+154h] [ebp-20Ch]
  char v7; // [esp+159h] [ebp-207h]
  __int16 v8; // [esp+255h] [ebp-10Bh]
  char v9; // [esp+257h] [ebp-109h]
  int v10; // [esp+258h] [ebp-108h]
  CHAR String; // [esp+25Ch] [ebp-104h]
  char v12; // [esp+25Fh] [ebp-101h]
  char v13; // [esp+260h] [ebp-100h]
  char v14; // [esp+261h] [ebp-FFh]

  memset(&String, 0, 0x104u);
  v10 = 0;
  if ( a2 == 16 )                               // 0x0010表示WM_CLOSE，当一个窗口或应用程序要关闭时发送一个信号
  {
    DestroyWindow(hWnd);                        // 销毁当前的窗口
    PostQuitMessage(0);                         // 通知系统当前有一个线程发送了进程中止退出请求
  }
  else if ( a2 == 273 )                         // 0x0111表示WM_COMMAND,当用户选择一条菜单命令项或当某个控件发送一条消息给它的父窗口，一个快捷键被翻译
  {
    if ( a3 == 1000 )                           // 0x3E8表示WM_DDE_EXECUTE,一个DDE客户程序提交此消息给一个DDE服务程序来发送一个字符串给服务器让它象串行命令一样被处理，服务器通过提交WM_DDE_ACK消息来作回应;
    {
      GetDlgItemTextA(hWnd, 1002, &String, 260);// HWND指定对话框句柄，nID指定了要获取其标题的控件的整数标识符1002。lpStr指向要接收控件的标题或文本的缓冲区&String。nMaxCount指定了要拷贝到lpStr的字符串的最大长度260
      strlen(&String);
      if ( strlen(&String) > 6 )                // 输入的字符串长度大于6，直接中止进程退出
        ExitProcess(0);
      v10 = atoi(&String) + 1;                  // 把输入的字符串转换成整型数
      if ( v10 == 123 && v12 == 120 && v14 == 122 && v13 == 121 )// 要求转换的v10是123，后面的字符v12是'x'，v13是'y'，v14是'z'
      {
        strcpy(Text, "flag");
        memset(&v7, 0, 0xFCu);
        v8 = 0;
        v9 = 0;
        _itoa(v10, &v5, 10);                    // itoa()函数取整数输入值，并将其转换为10进制数字的字符串。v10是int型整数123，所以&v5的值就是字符串“123”
        strcat(Text, "{");                      // 字符串拼接成flag
        strcat(Text, &v5);
        strcat(Text, "_");
        strcat(Text, "Buff3r_0v3rf|0w");
        strcat(Text, "}");
        MessageBoxA(0, Text, "well done", 0);   // 弹出新的对话框，内容是Text，标题是"well done"
      }
      SetTimer(hWnd, 1u, 0x3E8u, TimerFunc);
    }
    if ( a3 == 1001 )
      KillTimer(hWnd, 1u);
  }
  return 0;
}
```

重点的代码是：

```
memset(&String, 0, 0x104u);
v10 = 0;
if ( a2 == 16 )                         // 0x0010表示WM_CLOSE，当一个窗口或应用程序要关闭时发送
{
  DestroyWindow(hWnd);                  // 销毁当前的窗口
  PostQuitMessage(0);                   // 通知系统当前有一个线程发送了进程中止退出请求
}
else if ( a2 == 273 )                   // 0x0111表示WM_COMMAND,当用户选择一条菜单命令项或当某
{
  if ( a3 == 1000 )                     // 0x3E8表示WM_DDE_EXECUTE,一个DDE客户程序提交此消息给
  {
    GetDlgItemTextA(hWnd, 1002, &String, 260);// HWND指定对话框句柄，nID指定了要获取其标题的控件的整数
    strlen(&String);
    if ( strlen(&String) > 6 )          // 输入的字符串长度大于6，直接中止进程退出
      ExitProcess(0);
    v10 = atoi(&String) + 1;            // 把输入的字符串转换成整型数
    if ( v10 == 123 && v12 == 120 && v14 == 122 && v13 == 121 )
    {
      strcpy(Text, "flag");
      memset(&v7, 0, 0xFCu);
      v8 = 0;
      v9 = 0;
      _itoa(v10, &v5, 10);
      strcat(Text, "{");
      strcat(Text, &v5);
      strcat(Text, "_");
      strcat(Text, "Buff3r_0v3rf|0w");
0000116D sub_401090:30 (40116D)
```



```
v10 = atoi(&String) + 1;
if ( v10 == 123 && v12 == 120 && v14 == 122 && v13 == 121 )
```

if判断要求转换后的v10是123，atoi(表示 ascii to integer)是把字符串转换成整型数的一个函数，那么输入的字符串前3个字符就是“122”；而后面的字符v12是’x’（120），v13是’y’（121），v14是’z’（122）。
在上面的变量声明中，我们可以看到String、v12、v13、v14的地址：

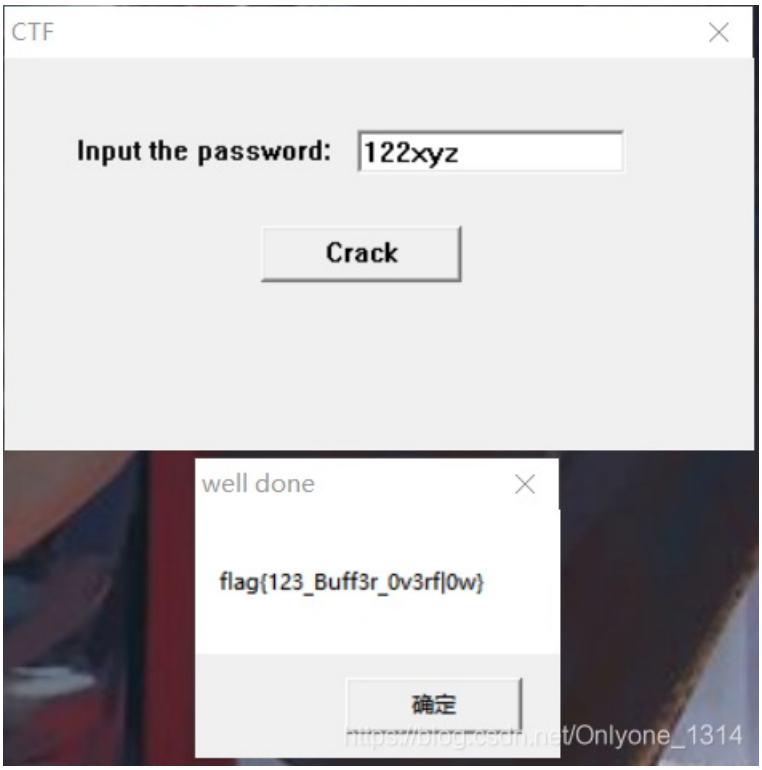
```
int __stdcall sub_401090(HWND hWnd, int a2, int a3, int a4)
{
  char v5; // [esp+50h] [ebp-310h]
  CHAR Text[4]; // [esp+154h] [ebp-20Ch]
  char v7; // [esp+159h] [ebp-207h]
  __int16 v8; // [esp+255h] [ebp-10Bh]
  char v9; // [esp+257h] [ebp-109h]
  int v10; // [esp+258h] [ebp-108h]
  CHAR String; // [esp+25Ch] [ebp-104h]
  char v12; // [esp+25Fh] [ebp-101h]
  char v13; // [esp+260h] [ebp-100h]
  char v14; // [esp+261h] [ebp-FFh]

  memset(&String, 0, 0x104u);
  v10 = 0;
  if ( a2 == 16 )                                   // 0x0010表示WM_CLOSE，当一个窗口或
  {
```

其中String的起始地址是[esp+25Ch]，v12在String起始地址后面3个字节的[esp+25Fh]，v13在v12起始地址后面1个字节的[esp+260h]，v14在v13起始地址后面1个字节的[esp+261h]；

所以我们输入到&String的字符串，第4个字节就是v12的值，第5个字节就是v13的值，第6个字节就是v14的值；

从而我们输入的字符串应该是“122xyz”，这样转换到v10的值就是123，后面的“xyz”不是10进制以内atoi函数会中断舍弃；得到v12的值就是‘x’,v13的值就是‘y’,v14的值就是‘z’,从而通过`if ( v10 == 123 && v12 == 120 && v14 == 122 && v13 == 121 )`判断：



得到正确的flag：`flag{123_Buff3r_0v3rf|0w}`。