LAB6实验报告

专业：信息安全 学号：2011395 姓名：魏伯繁

### ***一、实验工具：***

IDAPro7.0 IDAPro7.7 win10虚拟机 winxp虚拟机

### ***二、实验环境：***

静态分析主要在win10虚拟机完成。

内存4GB

主频2.9GHZ

### ***三、实验内容***

第一部分：LAB6-1

1、由main函数调用的唯一子过程的中发现的主要代码结构是什么？

**是一个if语句，用来判断主机是否正常联网**

2、位于0X40105F的子过程是什么？

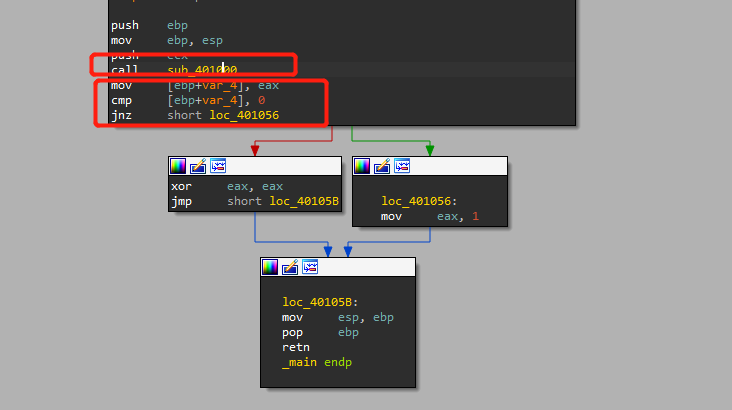
**可以推断出，该子过程完成了printf的任务**

3、这个程序的目的是什么？

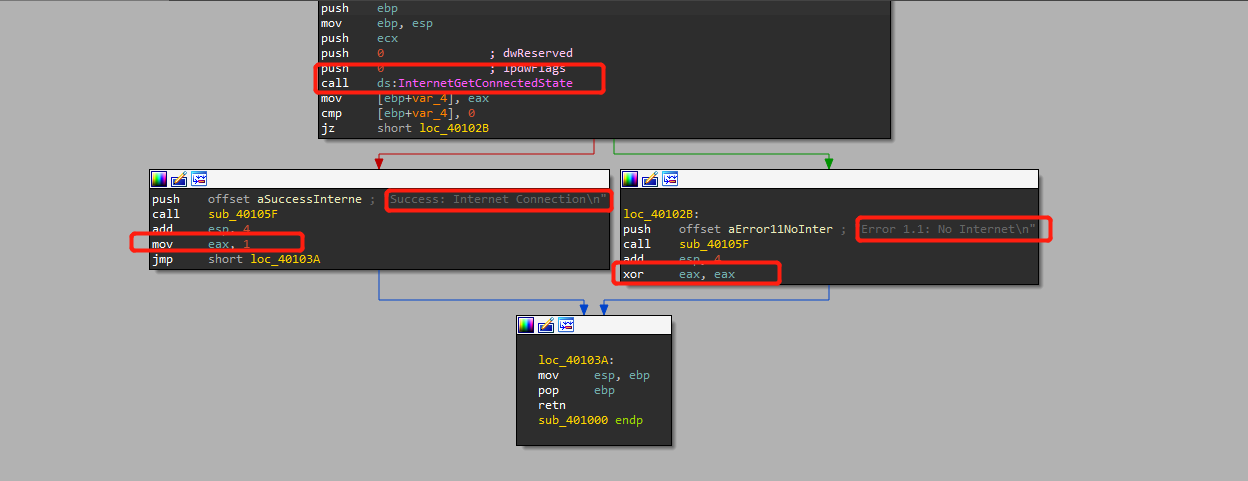
**检测主机是否存在一个可用的internet连接，如果有返回1并打印相应字符串，如果没有就返回0并打印相应字符串**

分析过程：

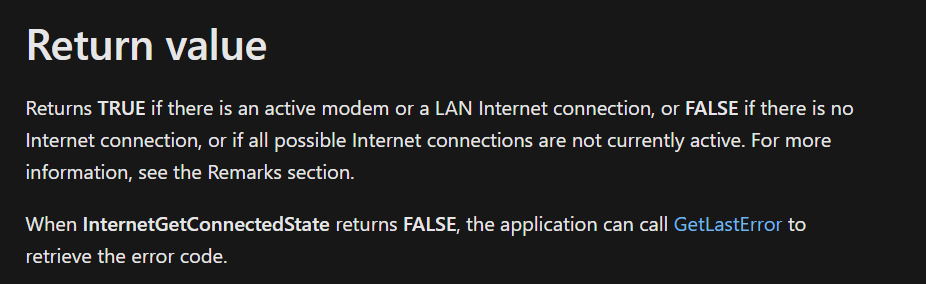
首先查看main函数的结构图：发现main函数调用了一个函数，并且根据这个函数的返回值进行了不同处理，如果这个函数的返回值是0就顺序执行并通过与自身异或使eax为零并返回，如果不是零就跳转到loc\_401056并最终返回1，



子函数的内容：子函数的核心也是调用了一个函数就是InternetGetConnectedState，这个函数的作用在下面阐释了，就是返回一下这个主机是否进行了连接互联网，如果返回是true就打印一个字符串Success：Internet Connection，反之就发音Error 1.1：no internet，并且，我们可以看到这两个字符串都进行了压栈，并且最后都跟随了换行符，那么我们可以猜测，这两个字符串是以参数形式进行的压栈，那么下一个函数的调用极大概率是一个字符串输出的工作，也就是printf

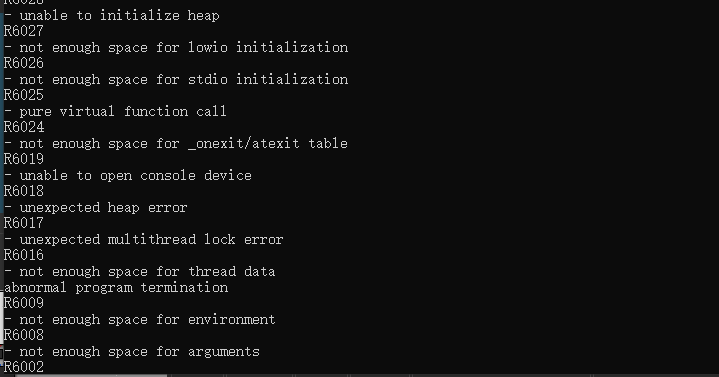


通过查看windows官方文档确认这个函数的作用：如果有一个可用的LAN连接就返回true，反之就返回false



最后，我们依然使用静态分析的方法再次对exe文件进行检测：

该恶意代码会反馈非常详细的错误信息，我们可以猜测，这些信息可能都是要返回给攻击者的，那么查看他的导入函数，也包括很多和网络相关的API函数，那么我们可以基本判定这个恶意代码是为了检测主机是否进行了网络连接，他可能是一大段恶意代码的一小部分，用来进行环境检测并决定后续的攻击方式。





第二部分：LAB6-2

1、main函数的第一个子过程执行了什么操作

**判断是否存在网络连接**

2、位于0x40117F的子过程是什么

**printf，打印文本的操作**

3、被main函数调用的第二个子过程做了什么？

**加载恶意网站<http://www.practicalmalwareanalysis.com>**

4、在这个子过程中使用了什么类型的代码结构？

**使用了一个大小为200的字符型数组**

5、在这个程序中有什么基于网络的暗示吗

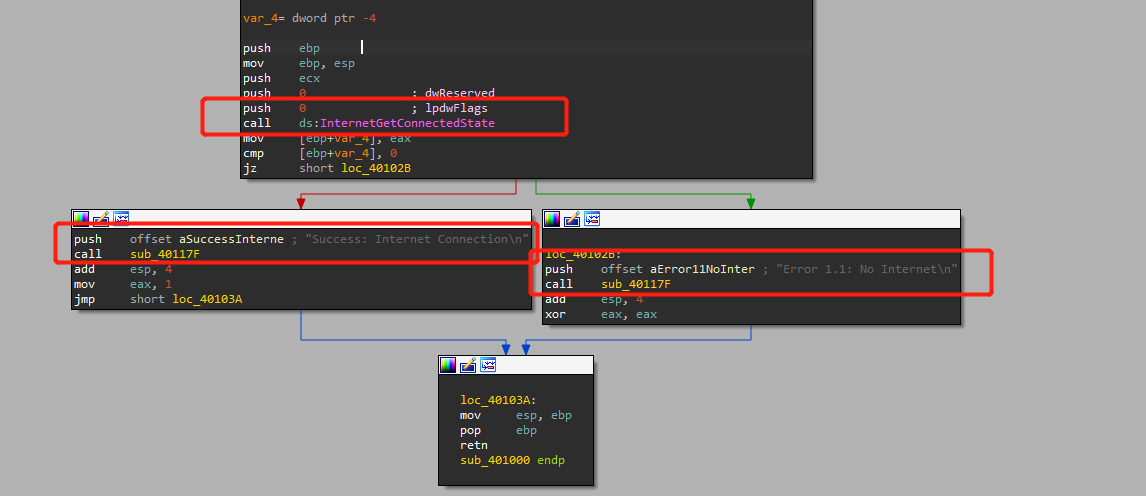
**当然有，首先他判断了是否具备网络连接，还是用Internet Exploer浏览器，并制定了URL的具体内容<http://www.practicalmalwareanalysis.com>，并且加载了网页内容**

6、这个恶意代码的目的是什么？

**他的终极目的就是从要加载的恶意网站上的html中取出指定的字符，但是要办成这样的目的需要他在之前做很多工作，比如说要先检测是否能连接网络，如果能则访问恶意网站，并且从恶意网站中下载html文件的内容，如果其开头以标准注释开头我就读取后面的字符内容并输出**

分析过程：

首先，前两个问题其实和第一问是一样的，其中第一个子过程判断了是否能够连接到网络，0x40117F的子过程完成了printf的工作。

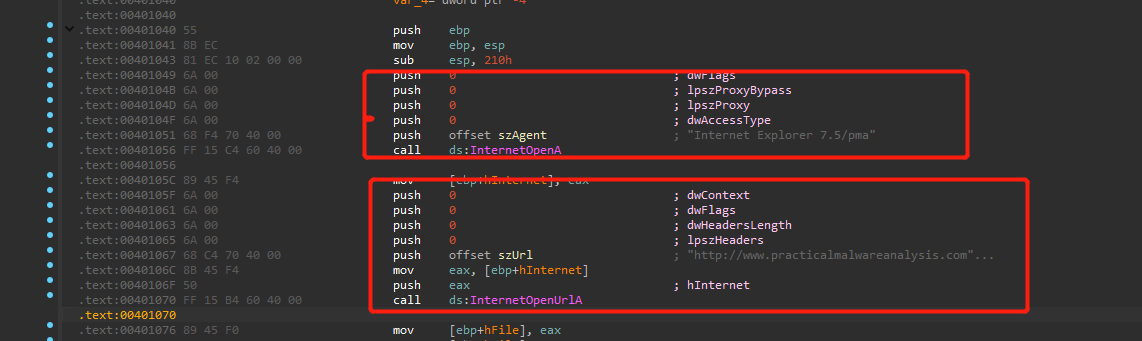


但是在这个exe文件中还有第二个子过程，点开这个子过程查看其内容，这个子过程首先调用了InternetOpenA以及InternetOpenUrlA

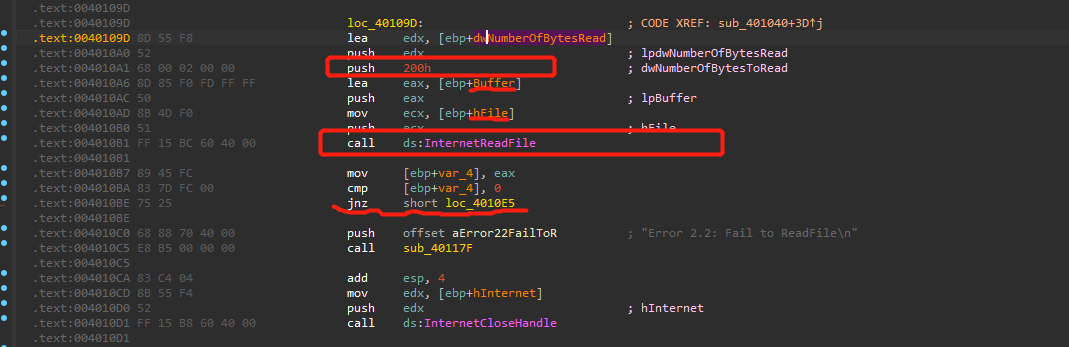
通过查阅windows官方文档，我们得知InternetOpenA的含义是：

**InternetOpen** 是应用程序调用的第一个 WinINet 函数。 它告知 Internet DLL 初始化内部数据结构，并为应用程序未来的调用做好准备。 应用程序使用 Internet 函数后，应调用 [InternetCloseHandle](https://learn.microsoft.com/zh-cn/windows/desktop/api/wininet/nf-wininet-internetclosehandle) 以释放句柄和任何关联的资源。而作为参数传入的字符串Internet Explorer 7.5/pma看起来像是一个microsoft推荐的浏览器类型。

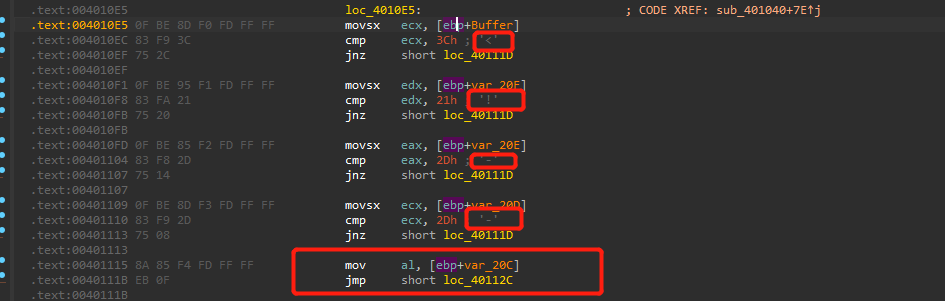
InternetOpenUrlA的作用则显而易见，根据官方文档的定义，该函数用来：打开由完整的 FTP 或 HTTP URL 指定的资源。如果成功建立连接，则返回 URL 的有效句柄;如果连接失败，则返回 **NULL**，最后的句柄存储在了eax中。



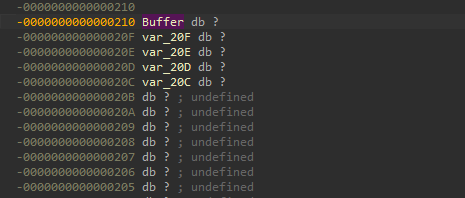
取出句柄之后，会判断这个句柄是否为空，如果为空就跳转到错误处理函数，如果不为空就继续下一步，下一个部分它调用了InternetReadFile函数，这个函数的含义从字面也可以了解，官方文档对他的描述是：从 [InternetOpenUrl](https://learn.microsoft.com/en-us/windows/desktop/api/wininet/nf-wininet-internetopenurla)、[FtpOpenFile](https://learn.microsoft.com/en-us/windows/desktop/api/wininet/nf-wininet-ftpopenfilea) 或 [HttpOpenRequest](https://learn.microsoft.com/en-us/windows/desktop/api/wininet/nf-wininet-httpopenrequesta) 函数打开的句柄读取数据。如果成功，则返回 **TRUE**，反之则返回**FALSE**。并且将buffer作为接受数据的缓冲区指针，并且指定了读入的字节数为200字节。如果最后读取的不成功就去错误处理的地方继续执行，如果最后成功返回了那就继续执行下一个阶段：



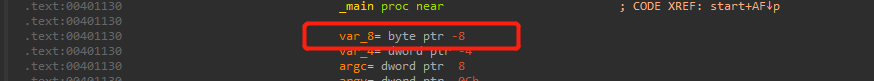
在下一个阶段，很明显的进行了一系列的比较操作，比较的内容就是<!--，这很明显是一个在html中的注释操作，注释中的内容在编译html文件时是不会被展示的，如果buffer中的前4个字节都成功与目标字符串匹配，那么就进入下一个阶段的执行，下一个阶段只有一个目的就是把<!--后面的下一个字符的内容返回给al，使其作为返回值的一部分进行返回



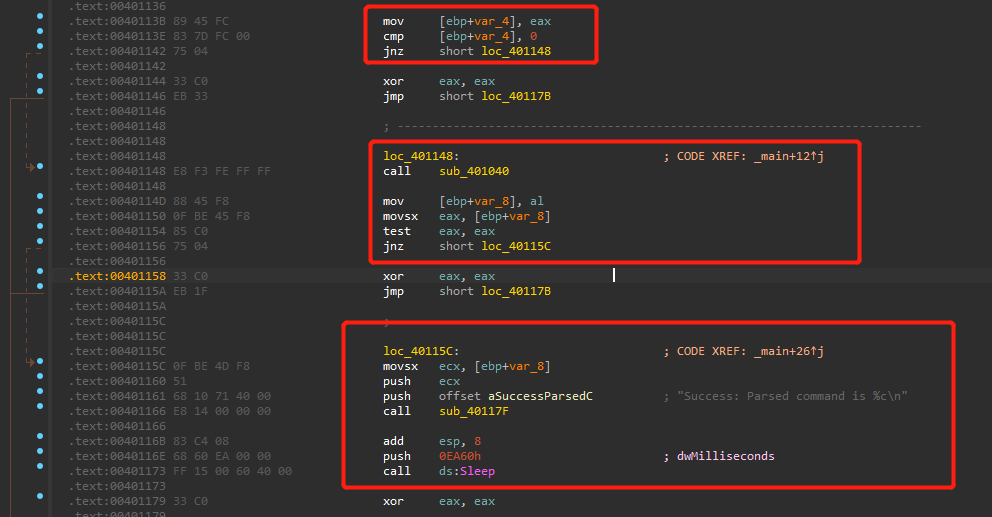
可以查看buffer的定义如下：



然后，我们返回主函数，在这一系列的调用之后，我们终于取出了想要取出的字符串，字符串的内容已经存在al中被返回，那么在主函数中还有一个定义的临时变量，在代码中将al的值赋值给了该临时变量，同样，如果什么都没取出来那么就进入到错误处理判断的函数位置，反之则正确输出他“Success parsed command”+字符



然后，我们通过printf并设置对应的格式化输出为%c来进行输出，这一点在前面一段的分析中已经说明，在输出结束后，我们发现了一个sleep调用，它的参数就是sleep的毫秒数，用16进制表示:转换后就是60000也就是休眠60秒，然后再退出





第三部分：LAB6-3：

1、比较在main函数中与实验6-2的main函数的调用，从main函数中调用的新的函数是什么

**新增的函数是0x401130处的函数，这个新函数中对从html中获取到的字符进行解析**

2、这个新函数的使用的参数是什么

**使用了两个参数，第一个参数是从html中解析的字符，第二个是argv[0]也就是程序名Lab06-03.exe**

3、这个函数包含的主要代码是什么

**包含了对于解析的字符串的选择，一个由跳转表实现的switch语句**

4、这个函数能够做什么

**这个函数能够根据传入的字符不同选择不同的动作，传入a就是创建文件夹，传入b就是复制文件，传入c就是删除文件，传入d就是更改注册表，传入e就是休眠100秒**

5、这个恶意代码的本地特征是什么

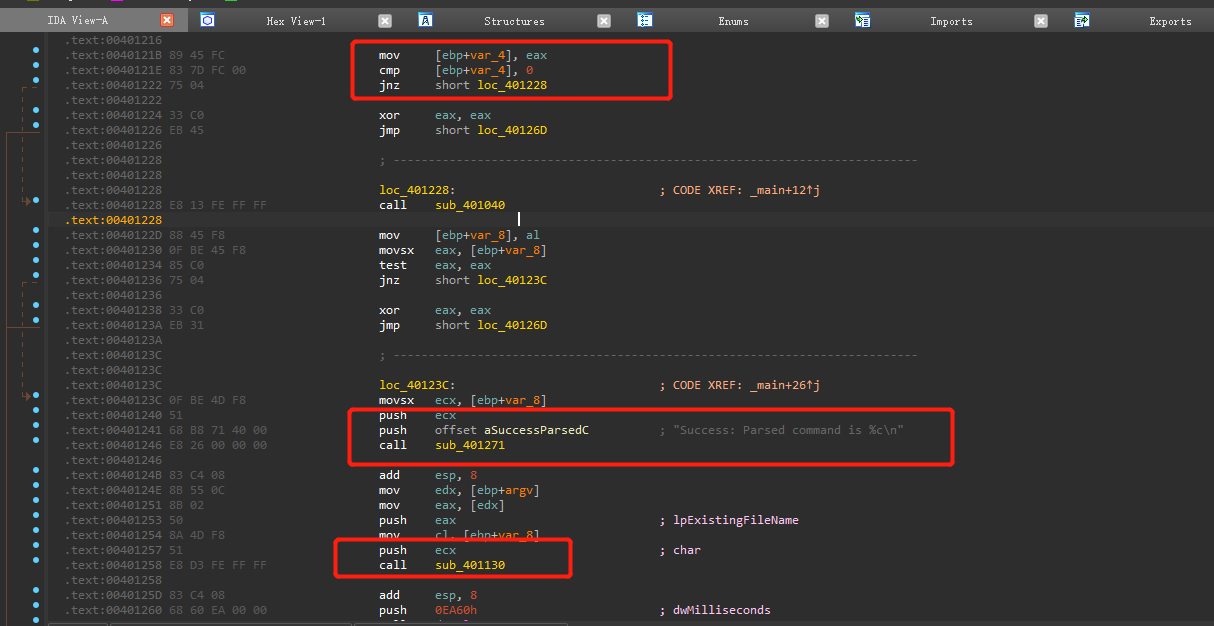
**本次特征包括C:\temp、C:\temp\cc.exe、SoftWare\Microsoft\Windows\CurrentVersion\Run\Malware**

6、这个恶意代码的目的是什么

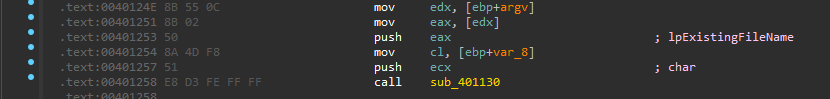
**这个恶意代码继承了Lab-06-02.exe的功能，先检测是否能连接网络，如果能则访问恶意网站，并且从恶意网站中下载html文件的内容，如果其开头以标准注释开头我就读取后面的字符内容并输出并且根据从html中获得的字符进行动作选择，包括的可能的动作有：传入a就是创建文件夹，传入b就是复制文件，传入c就是删除文件，传入d就是更改注册表，传入e就是休眠100秒**

分析过程：

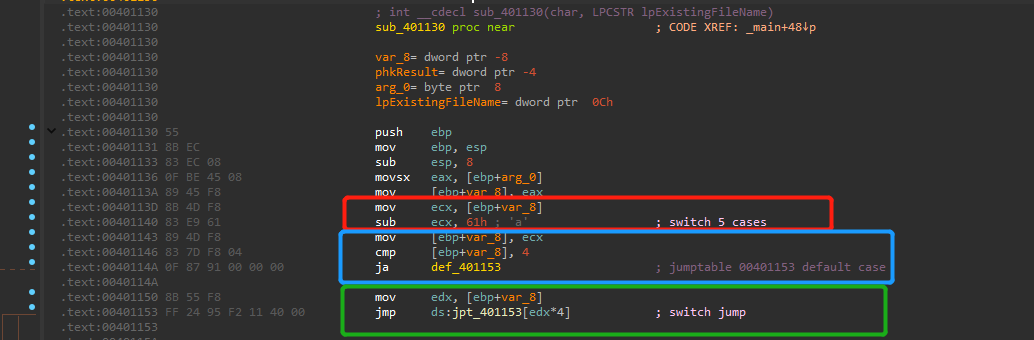
首先，main函数的大体内容和lab6-2中相差不到，首先调用的函数都是去判断是否能联网。取html、判断、取字符，然后输出那个字符是什么，mai函数中新添加的是一个函数sub\_401130



仔细分析对应的参数压栈过程：首先压入的第一个参数是从C语言右开始的第一个参数，也就是实际上我们在高级语言中通常说的第二个参数，他把argv的内容压入了栈中，其实就是这个程序的程序名。压入的第二个参数实际上是高级语言中我们通常说的第一个参数，他就是var\_8，也就是在lab6-2中我们分析的从html中取出的指定字符。



跳转到对应的子程序，我们不难发现该部分对应的代码其实是一个switch函数和一个跳转表，如果从html中取回的字符正确，应该是a、b、c、d、e中的一个，由于要符合跳转表的特性，代码将该值减去a的ASC码值对应跳转表中应该对应的项。为了保证程序的鲁棒性，代码对其他不应该出现的情况进行了判断，如果减出来这个值大于4，就跳转到错误处理地址（因为最多只有五个可能的值即0、1、2、3、4和5）然后根据最后减出来的这个值进行选择，其中edx中存了基地址



随后紧跟着的就是跳转表的内容，跳转表的合法功能总共有五个：

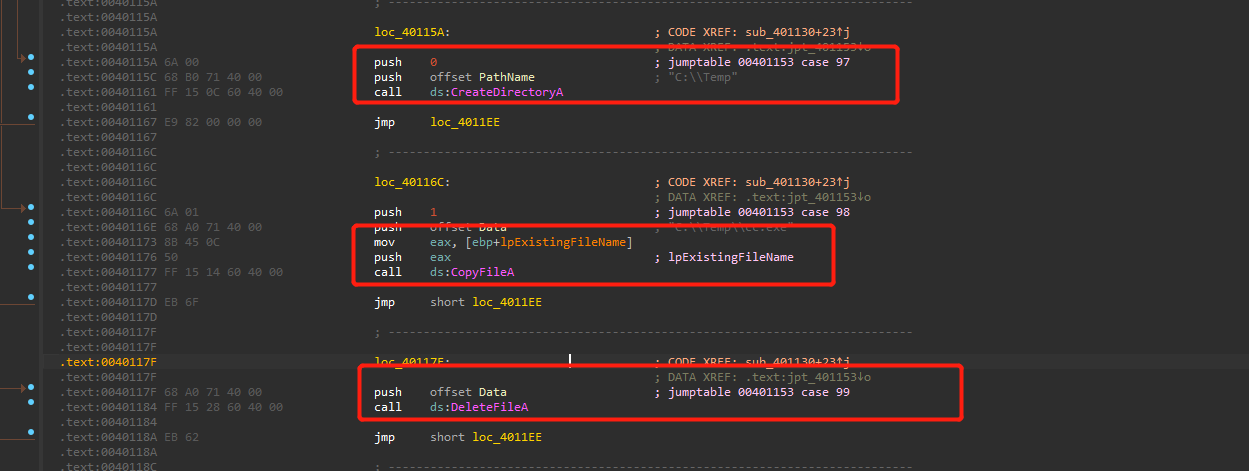
第一个是新建一个文件夹，如果C:\temp不存在那么就创建它。

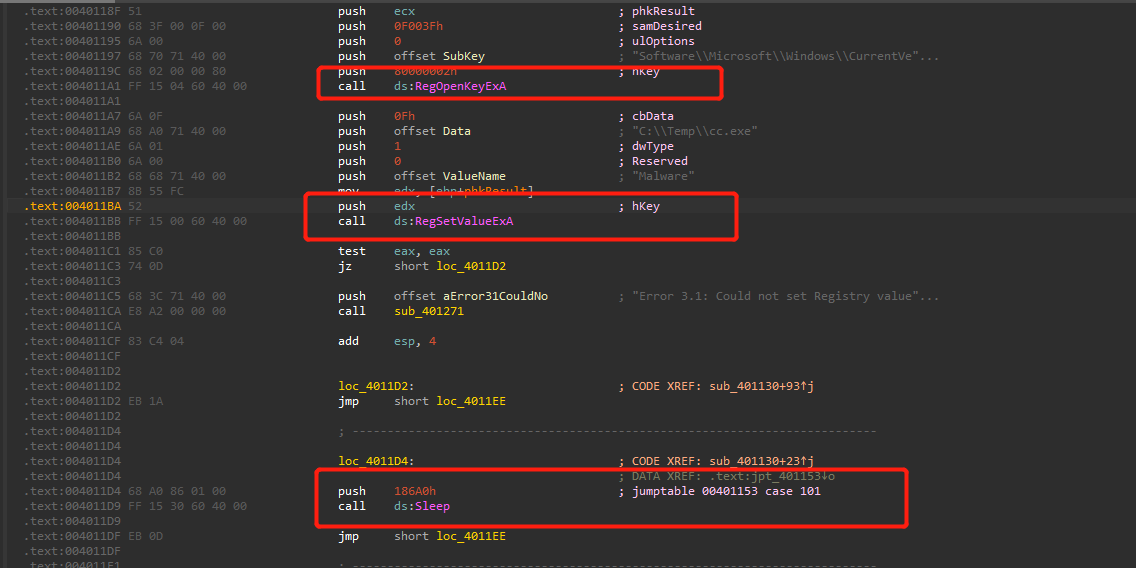
第二个复制一个文件，他想复制到的最终路径是C:\temp\cc.exe，他想复制的内容保存在了eax中，eax就是我们存进来的argv，他就是这个进程的进程名也就是lab06-03.exe，所以说这一段代码将正在运行的恶意代码复制为了C:\temp\cc.exe

第三个是删除一个文件，使用的参数一样是00401153，也就是说，如果C:\temp\cc.exe存在就删除他

第四个是修改注册表，将注册表SoftWare\Microsoft\Windows\CurrentVersion\Run\Malware的值设置为C:\temp\cc.exe这样的话每次系统启动时这个exe文件也会跟着进行自启动

第五个就是睡眠，睡眠的毫秒数用16进制表示：186A0，就是100秒







第四部分：Lab6-4

1、在实验6-3和实验6-4的main函数中的调用之间的区别是什么？

**区别不大，printf的地址更改了；00401000检查网络连接；00401040尝试进行远程网络连接；004010B1处进行网页数据爬取；004010F9进行字符串比较；00401150进行跳转表的选择实现**

2、什么新的代码结构被加入到了main中？

**加入了一个for循环来判断函数执行的次数，整个main函数将被循环1440次**

3、这个实验的解析html的函数和前面的实验中的那些函数有什么区别？

**在解析式还要加入现在整个程序已经循环了多少遍了，用格式化字符串%d作为占位符，具体参数由ecx提供：Internet Exploer 7.50/pma%d**

4、这个程序会运行多久（假设它成功接入了互联网）

**main函数没运行一次要休眠1分钟，main函数会执行1440编也就是会休眠总共1440分钟，假设程序处理其他流程（如网络数据检查爬取）的时间忽略不计，会执行24个小时**

5、在这个恶意代码中有什么新的基于网络的迹象吗

**Internet Exploer 7.50/pma%d就是新的迹象，一个新的参数被提供了**

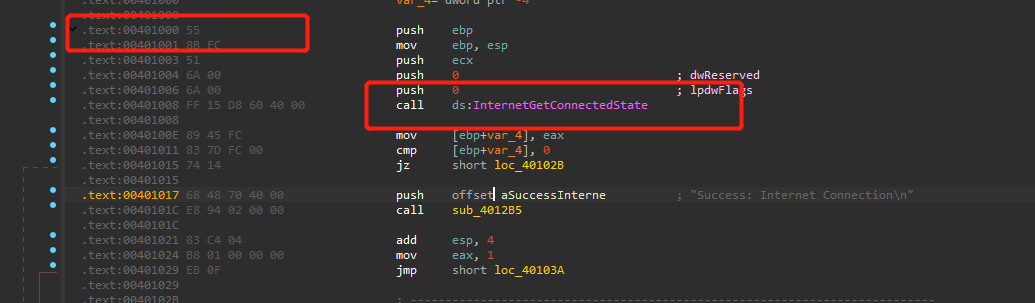
6、这个恶意代码的目的是什么

**这个恶意代码继承了Lab-06-02.exe的功能，先检测是否能连接网络，如果能则访问恶意网站，并且从恶意网站中下载html文件的内容，如果其开头以标准注释开头我就读取后面的字符内容并输出并且根据从html中获得的字符进行动作选择，包括的可能的动作有：传入a就是创建文件夹，传入b就是复制文件，传入c就是删除文件，传入d就是更改注册表，传入e就是休眠100秒，并且这个过程会总共进行1440次，也就是说，我可以修改网站的传入的参数值来控制被植入病毒的计算机执行我想要的动作并且多次执行**

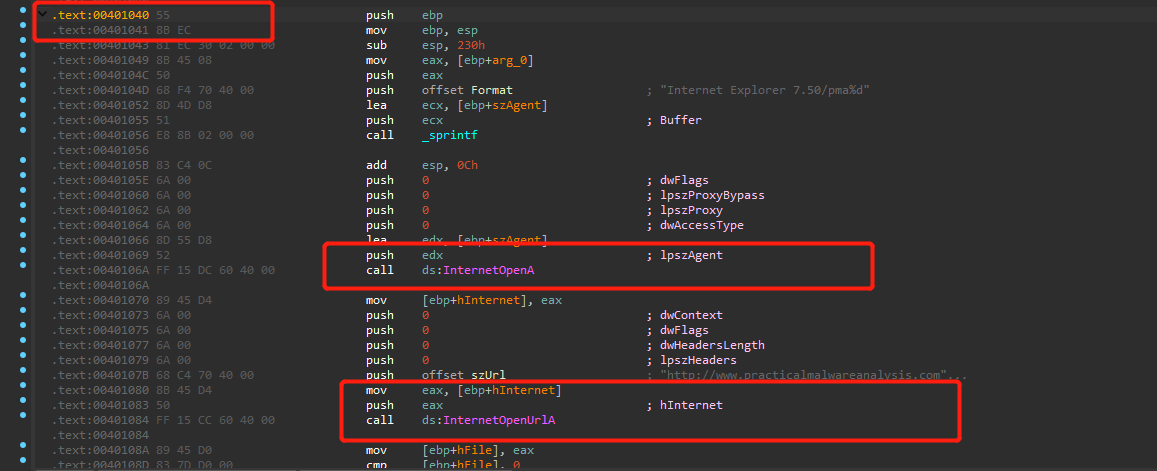
分析过程：

有了前面几次实验的铺垫，我大体清除了这个实验的结构是什么样子的，掌握了其大体的流程。其具体的函数调用和第三个可执行文件并没有什么区别，流程如下：

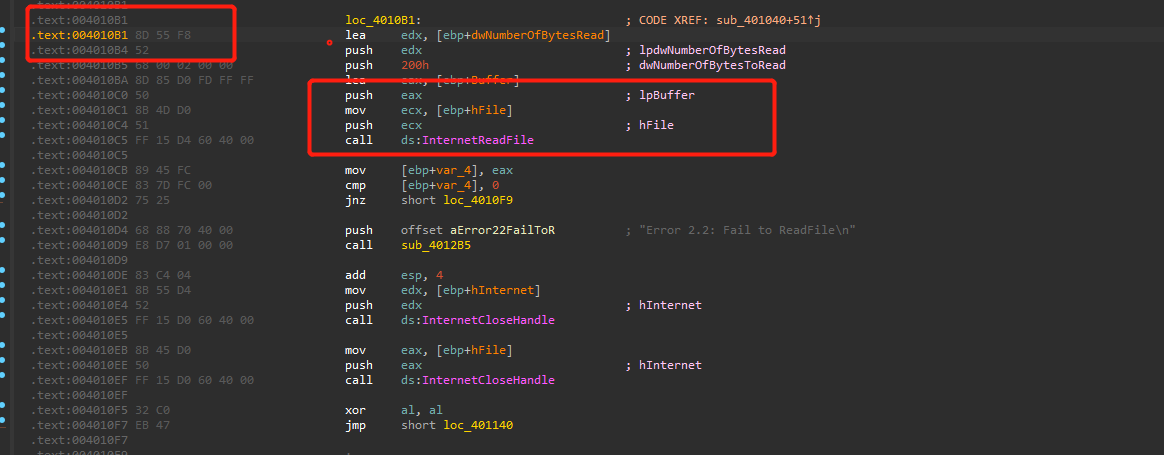
首先检查是否存在网络连接：在函数00401000处完成



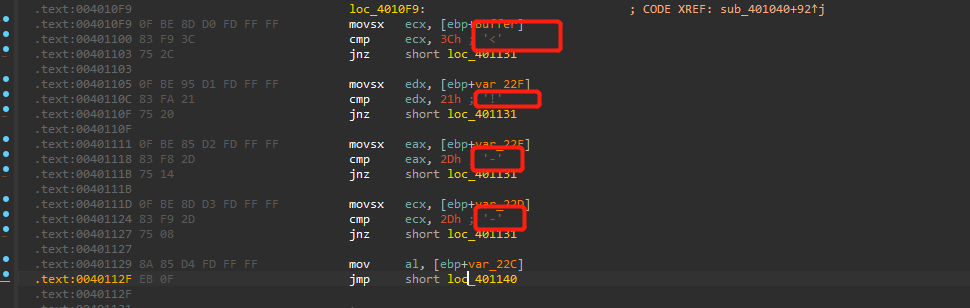
如果有网络连接就进行远程网络的连接：在函数00401040处完成



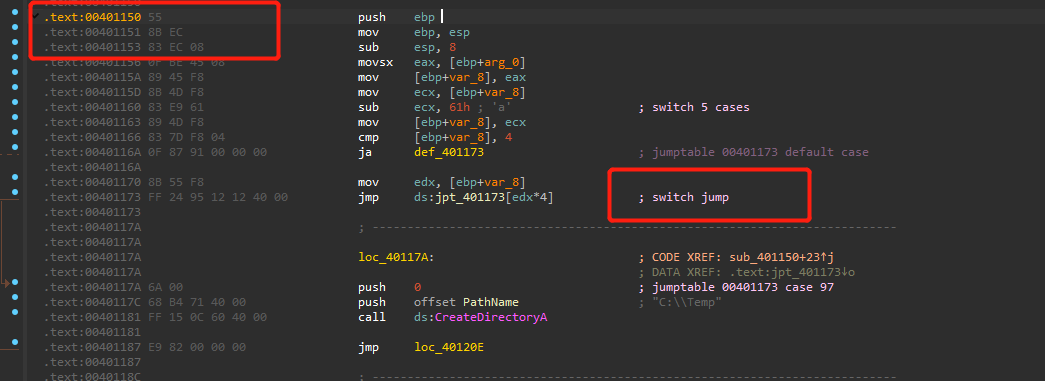
如果连接远程网络顺利那我们就读取网页数据，并与设定好的字符进行比较：在函数004010B1处完成



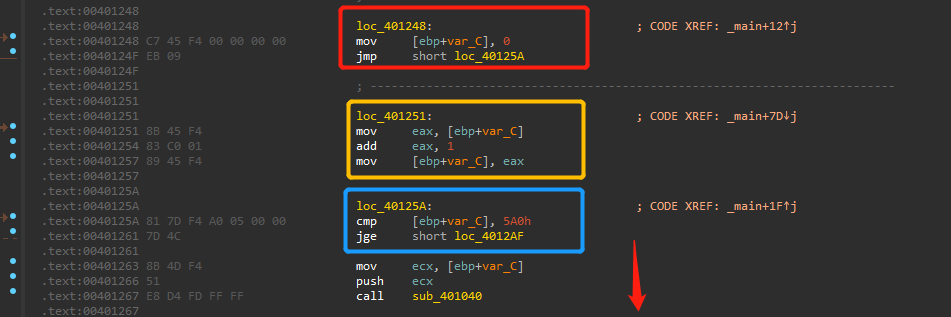
字符比较在004010F9完成，如果比较成功，将下一个字符传入al



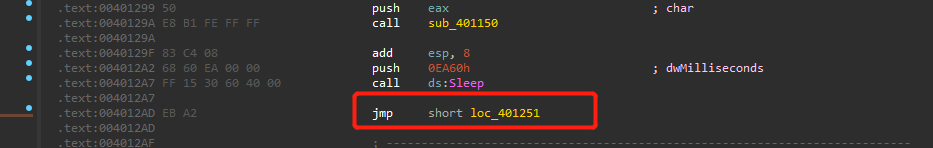
然后和第三次实验一致，进行一个switch语句以及跳转表来根据al中的值采取不同的动作



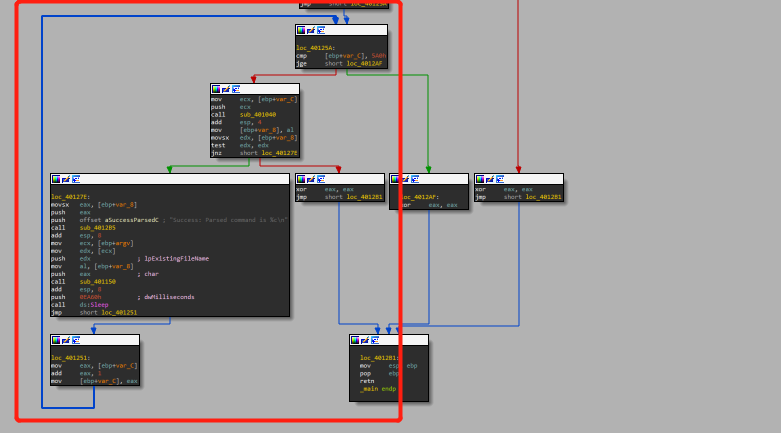
本次实验与上次实验不同的就在于main函数中多出了一个代码结构就是循环语句，for循环在00401248处进行了初始化，把VarC初始化为0，然后就跳转到0040125A处进行比较，其中5A0h是16进制数，翻译成10进制数是1440，如果判断varC的值不大于1440就进行后面的执行，否则跳转退出，最后for循环的自加操作在00401251处完成，每次进行加一操作，也就是说这个for循环会循环1440次



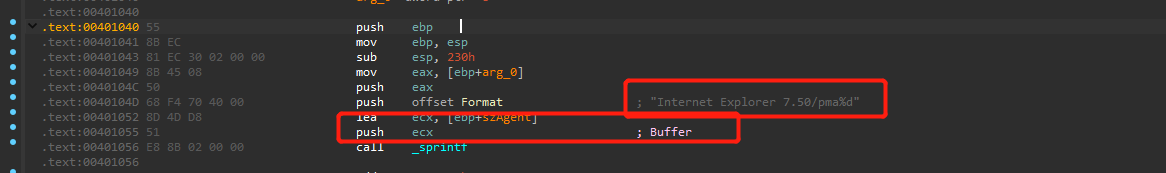
代码段结束，跳回到自加：



这么看这个for循环的结构会清晰一些：



最后，我们发现这个记录了循环次数的数值不仅被记录了，而且还被打印了，就在指定user-agent的部分将这个次数以格式化字符串%d的方式打印了出来，作为%d参数的是ecx也就是计数的寄存器。



第二部分：yara规则编写：

import "pe"
  
rule IsPE{
  
 meta:
  
 description = "检查文件是否为PE文件"
  
 author="wbf"
  
 date="2022-10-29"
  
 condition:
  
 uint16(0) == 0x5A4D and //“MZ”头
  
 uint32(uint32(0x3C)) == 0x00004550 // “PE”头
  
  
}
  
  
rule smallFileSize{
  
 meta:
  
 description = "检查文件小是否大概率为exe文件"
  
 author="wbf"
  
 date="2022-10-29"
  
 condition:
  
 filesize<500KB
  
}
  
  
rule mayBeMaleware{
  
 meta:
  
 description = "检查文件小是否大概率为maleware文件"
  
 author="wbf"
  
 date="2022-10-29"
  
 strings:
  
 $a = { E8 00 00 00 00 }
  
 $b = "malware" nocase
  
 condition:
  
 $a at pe.entry\_point or $b
  
}
  
  
rule WebConnect{
  
 meta:
  
 description = "该exe可能进行了网络连接"
  
 author="wbf"
  
 date="2022-11-3"
  
 strings:
  
 $a = "http://www.practicalmalwareanalysis.com/cc.htm"
  
 $b = "No Internet"
  
 $c = "Success: Internet Connection"
  
 $d = "Internet Explorer 7.50/pma"
  
 $e = "GetLastError"
  
 $f = "InternetOpenA"
  
 $g = "WININET.dll"
  
 $h = "InternetOpenUrlA"
  
 condition:
  
 (5 of them) and IsPE and smallFileSize
  
}
  
  
rule operateLocalFile{
  
 meta:
  
 description = "该软件可能更改本机配置"
  
 author="wbf"
  
 date="2022-10-29"
  
 strings:
  
 $a = "C:\Temp"
  
 $b = "FlushFileBuffers"
  
 $c = "RegSetValueExA"
  
 $d = "runtime error"
  
 condition:
  
 (2 of them) and IsPE and smallFileSize
  
}

第三部分：IDA PYTHON脚本编写

在本次实验中，很多地方都用到了查看函数，为了方便的获得一个函数的起始地址和结束地址可以编写IDA PYTHON脚本：

ea = here()
  
func = idaapi.get\_func(ea) # 获取地址所在函数的范围
  
print "Start: 0x%x, End 0x%x" % (func.startEA,func.endEA)

同样，我们可以获取一个函数部分的汇编代码：

ea = here()
  
start = idc.get\_func\_attr(ea, FUNCATTR\_START)
  
end = idc.get\_func\_attr(ea, FUNCATTR\_END)
  
cur\_addr = start
  
while cur\_addr <= end:
  
 print hex(cur\_addr), idc.generate\_disasm\_line(cur\_addr, 0)
  
 cur\_addr = idc.next\_head(cur\_addr, end)

识别库函数：（想在一开始printf中使用来的）

Python>for func in idautils.Functions():
  
 flags = idc.get\_func\_attr(func, FUNCATTR\_FLAGS)
  
 if flags & FUNC\_LIB:
  
 print hex(func), "FUNC\_LIB", idc.get\_func\_name(func)

由于exe中有很多jmp和call函数，可以编写脚本辅助分析：

python>for func in idautils.Functions(): # 获取已知函数list
  
 flags = idc.get\_func\_attr(func,FUNCATTR\_FLAGS)
  
 if flags & FUNC\_LIB or flags & FUNC\_THUNK:
  
 continue
  
 dism\_addr = list(idautils.FuncItems(func)) # 函数指令地址
  
 for line in dism\_addr:
  
 m = idc.print\_insn\_mnem(line)
  
 if m == 'call' or m == 'jmp':
  
 op = idc.get\_operand\_type(line,0)
  
 if op == o\_reg:
  
 print "0x%x %s" % (line,idc.generate\_disasm\_line(line,0))