**恶意代码分析与防治技术实验报告**

**Lab5**

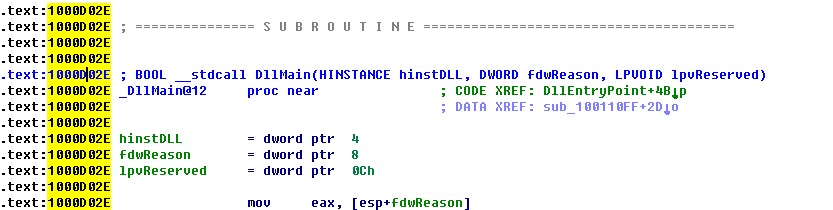
**学号：2013018 姓名：许健 专业：信息安全**

1. **实验内容**

使用IDA Pro分析在文件Lab05-01.dll中发现的恶意代码。在样本分析结果的基础上，编写样本的Yara检测规则。尝试编写IDA Python脚本来辅助样本分析。

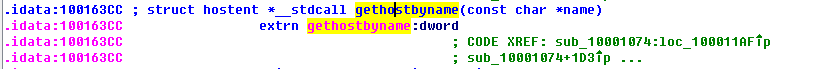
1. **问题解答**
2. DllMain的地址是什么?

使用IDA Pro加载恶意程序，可以直接来到位于**0x1000D02E**处的**DLLMain**函数，DLLMain是我们开始分析的地方。从DLLEntryPoint到DLLMain的函数是由编译器生成的，我们没必要分析。



1. 使用Imports窗口并浏览到gethostbyname, 导入函数定位到什么地址?

在导入表中找到并双击gethostbyname函数，转到反汇编代码，gethostbyname导入项驻留在文件.idata节的0x100163CC处。



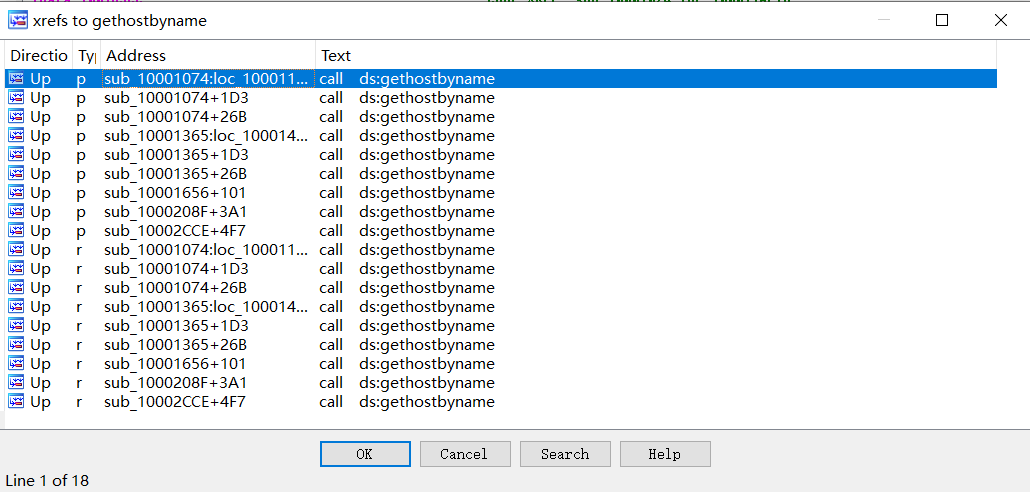
1. 有多少函数调用了gethostbyname?

光标保持在gethostbyname处，按**Crtl+X**键，可以检查它的交叉引用情况。

窗口最底下的“Line 1 of 18”表示存在对gethostbyname的9处交叉引用，这是因为IDA Pro会计算两次交叉引用：类型p是被“调用”的引用，类型r是被“读取”的引用。

（因为是对一个导入项call dword ptr[...]，因此CPU必须先读取这个导入项再调用它）

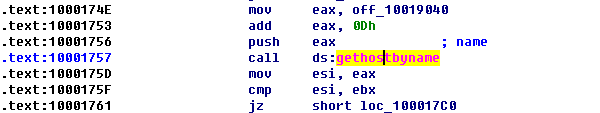
仔细检查交叉引用列表，发现在**5个不同的函数中**调用了gethostbyname。

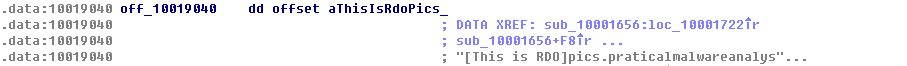


1. 将精力集中在位于0x10001757处的对gethostbyname的调用，你能找出哪个DNS请求将被触发吗?

我们可以快速定位到0x10001757，其中gethostbyname函数用到了一个参数，eax的值等于off\_10019400加上0Dh字节的偏移，跳转到该地址得知eax是一个指向**pics.practicalmalwareanalysis.com**域名字符串的指针。

这个调用会对该域名发起一个DNS请求，以获得其IP地址。

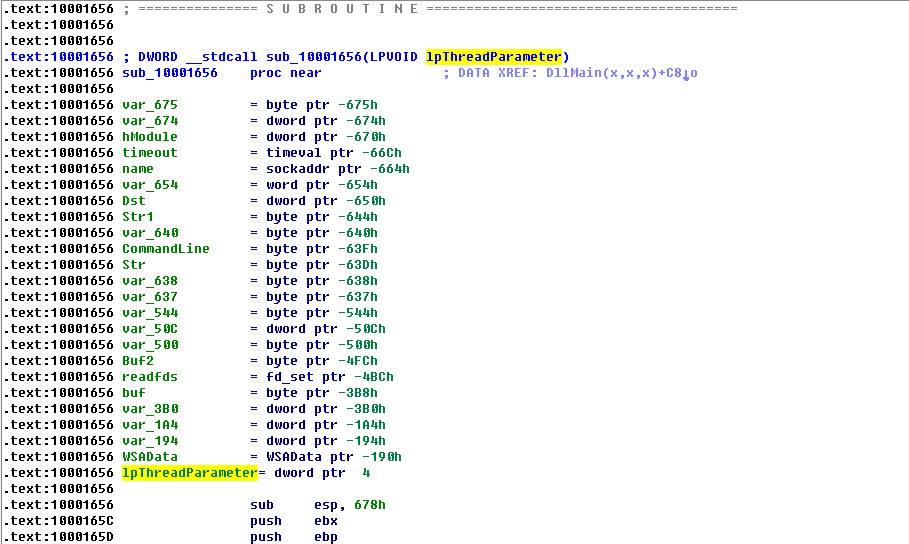




1. IDA Pro识别了在0x10001656处的子过程中的多少个局部变量?
2. IDA Pro识别了在0x10001656处的子过程中的多少个参数?

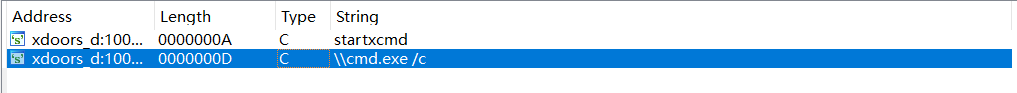
按G键来到0x10001656处，分析sub\_10001656的代码。

IDA Pro一共识别出23个局部变量（负偏移值）、一个函数的参数lpThreadParameter

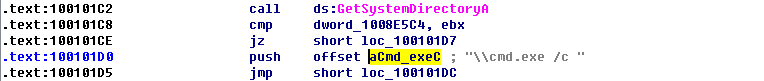


1. 使用Strings窗口，在反汇编中定位字符串\cmd.exe /c。它位于哪?
2. 在引用\cmd.exe /c的代码所在区域发生了什么?
3. 在同样的区域，在0x100101C8处，看起来好像dword\_1008E5C4是一个全局变量，它帮助决定走哪条路径。那恶意代码是如何设置dword\_1008E5C4的呢?（提示：使用dword\_1008E5C4的交叉引用）

在Strings窗口看到该字符串位于PE文件的xdoors\_d节的0x10095B34处。



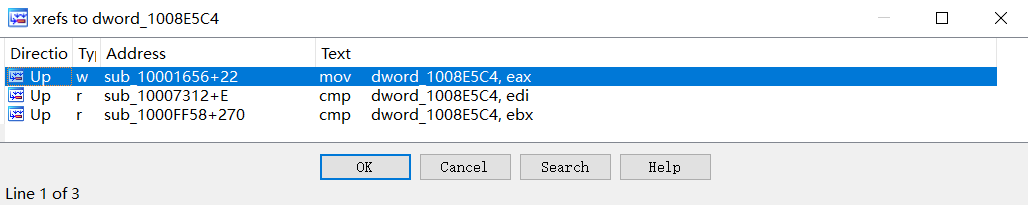
检查对该字符串的交叉引用，只看到一处位于0x100101D0，该字符串被压入栈中。



应该是用于远程会话。

查看函数sub\_1000FF58的图形模式，可以看到一系列的memcmp函数被用于比较如cd、exit、install、inject、uptime等字符串。还可以看到在0x1001009D处，有一个字符串This Remote Shell Session。查看这个函数和它发起的调用，可以看到一系列的recv和send调用，因此我们猜测这是一个远程shell会话函数。

查看全局变量dword\_1008E5C4的交叉引用，两处比较一处赋值，跳转到赋值的地方，可以看到函数sub\_10003695的返回值被赋给dword\_1008E5C4





查看sub\_10003695的反汇编代码，GetVersionEx用于获取当前操作系统的版本信息，然后根据dwPlatformId与2比较的结果设置al寄存器，如果当前操作系统是Windows 2000或更高版本，全局变量dword\_1008E5C4会被设置为1。

VersionInformation.dwPlatformId

Result:=OSVI.dwPlatformId;

{

返回值:

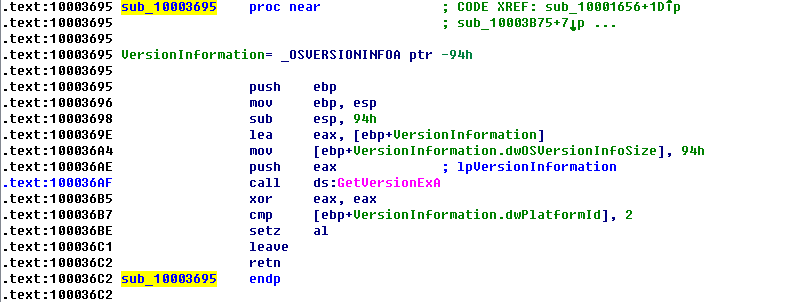
为0表示为win3x系统;

为1表示为win9x系统;

为2表示为winNT;

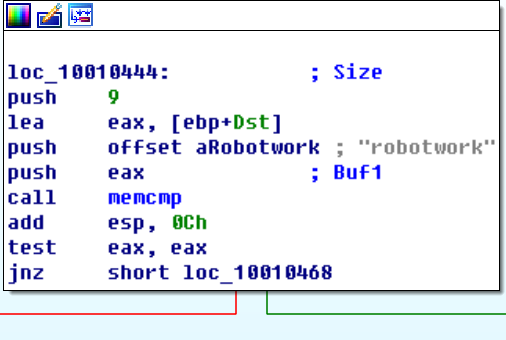
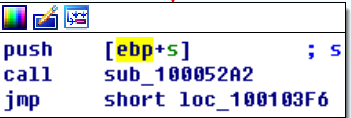
为3表示为win2000系统;

}



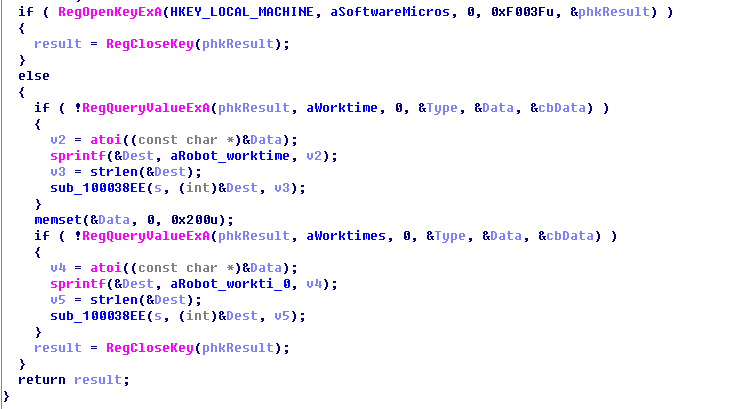
1. 在位于0x1000FF58处的子过程中的几百行指令中，有一系列使用memcmp来比较字符串的指令。如果对rotbotwork的字符串比较是成功的（memcmp返回0），会发生什么?

找到robotwork处的memcmp，如果字符串比较成功则jnz不会跳转，执行红色分支，也就是call sub\_100052A2。

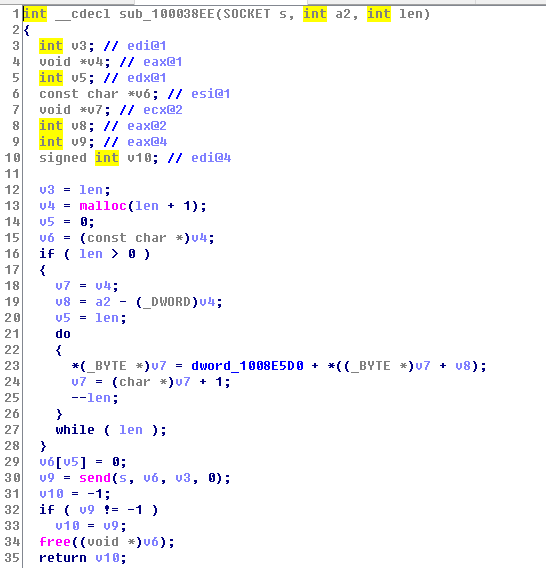


查看sub\_100052A2的代码，汇编代码看起来比较费力可以F5查看C代码，更容易理解代码的逻辑。



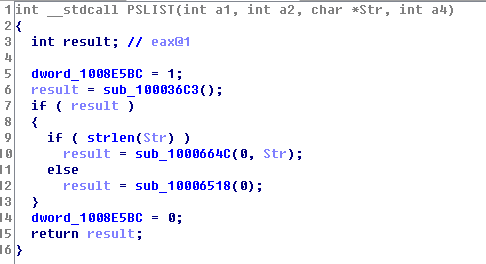
可以看到它查询了注册表HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTime和WorkTimes键的值，然后把这一消息返回给传给该函数的网络套接字。

sub\_100038EE调用了send函数，应该是用于在网络通信中发送消息。

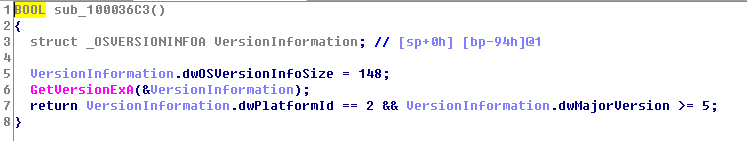


1. PSLIST导出函数做了什么?

定位导出函数PSLIST，查看C代码，可以看到它的分支取决于sub\_100036C3()的结果。

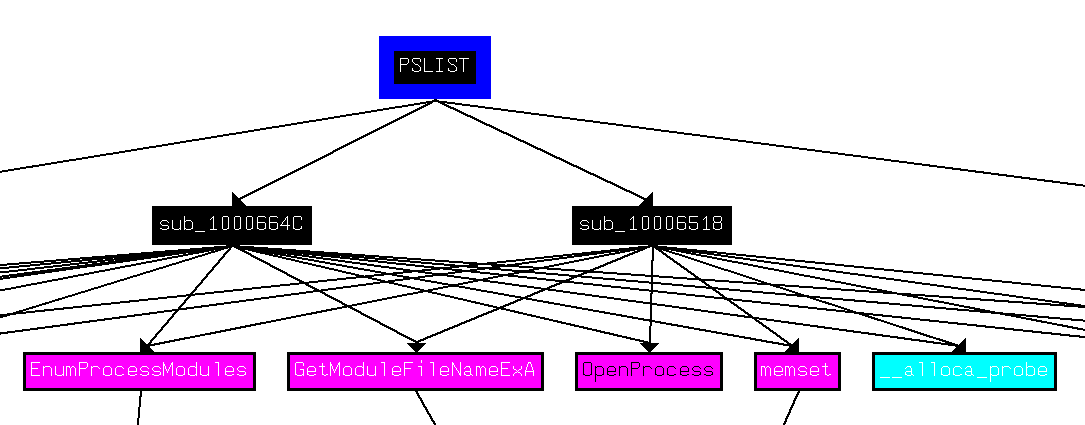


sub\_100036C3函数检查操作系统的版本是Windows Vista/7或是Windows Xp/2003/2000。



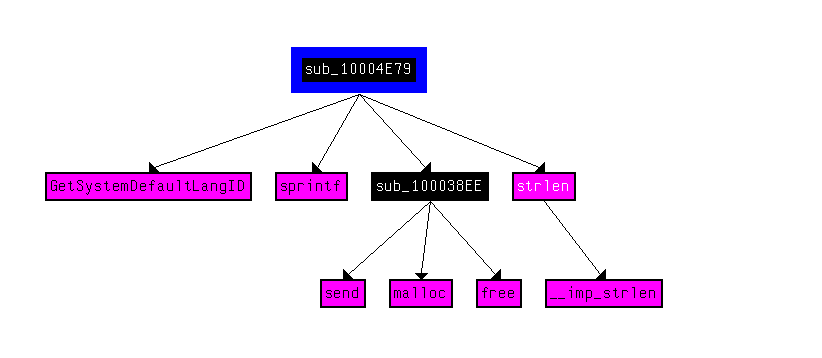
然后根据str的长度分别调用了sub\_1000664C和sub\_10006518函数，这两个函数都调用了 一些相同的函数，比如CreateToolhelp32Snapshot，可以获得系统中正在运行的进程、线程信息；EnumProcessModules，用于获取进程的地址。从调用的API来看，应该是要获得一个进程列表。





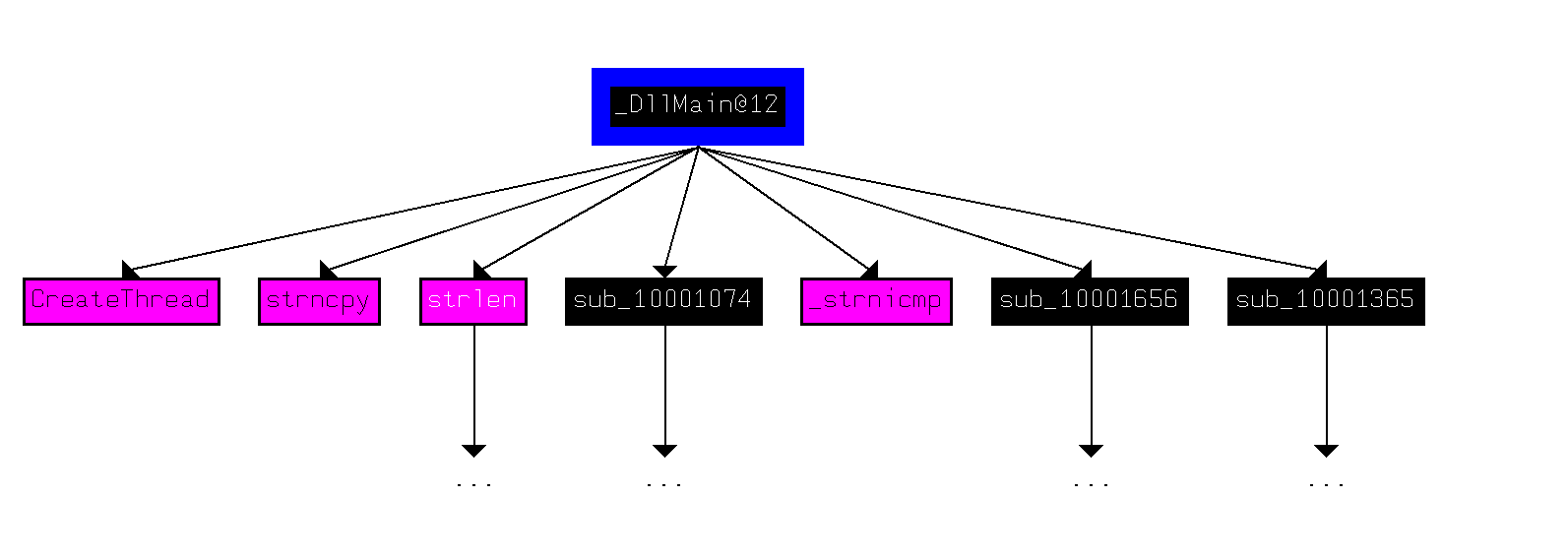
1. 使用图模式来绘制出对sub\_10004E79的交叉引用图。当进入这个函数时，哪个API函数可能被调用? 仅仅基于这些API函数，你会如何重命名这个函数?

View->Graph->Xrefs from，可以得到函数的交叉引用图，可以看到该函数调用了GetSystemDefaultLangID和send。这一信息告诉我们该函数可能可以通过socket发送语言标志，因此可以给他取名为send\_LangID。

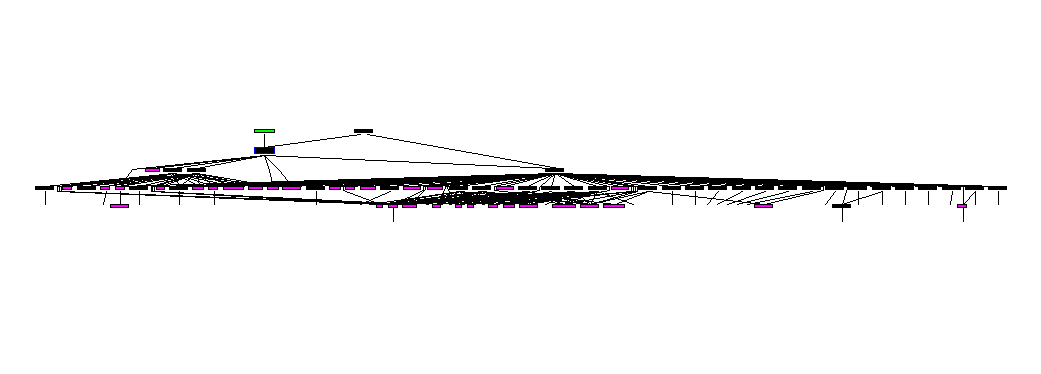


1. DllMain直接调用了多少个Windows API? 多少个在深度为2时被调用?

使用用户自定义交叉引用，设置递归深度为1，可以看到直接调用的Windows API

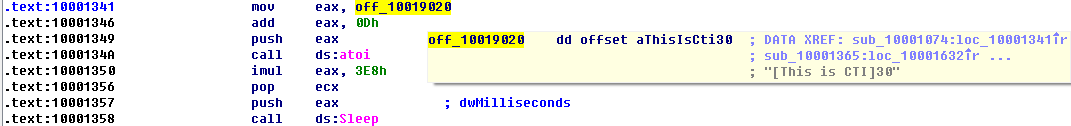


设置递归深度为2，显示所有被调用函数。看到的图就已经十分庞大，还有回调DLLmain的函数，调用的API函数非常多。



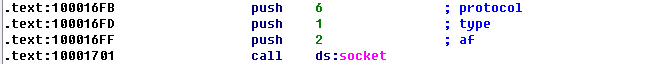
1. 在0x10001358处，有一个对Sleep（一个使用包含要睡眠的毫秒数的参数的API函数）的调用。顺着代码向后看，如果这段代码执行，这个程序会睡眠多久?

Sleep函数的参数是压入栈的EAX寄存器，往回看分析EAX寄存器的值，它指向了字符串30，然后转换成了数字再乘以0x3E8h（1000），得到30000毫秒，也就是30秒。也就是程序休眠的时间。



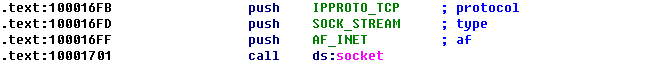
1. 在0x10001701处是一个对socket的调用。它的3个参数是什么?

使用的是IPPROTO\_TCP协议，类型是SOCK\_STREAM，IP地址是AF\_INET，用于设置IPv4 socket



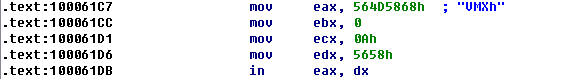
1. 使用MSDN页面的socket和IDA Pro中的命名符号常量，你能使参数更加有意义吗? 在你应用了修改以后，参数是什么?

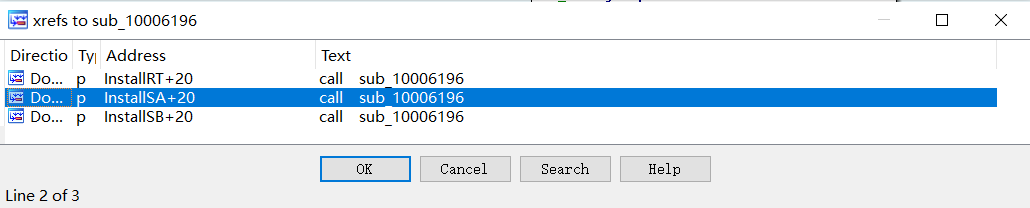
对socket调用使用符号常量后的显示，可以清楚每个参数的含义。

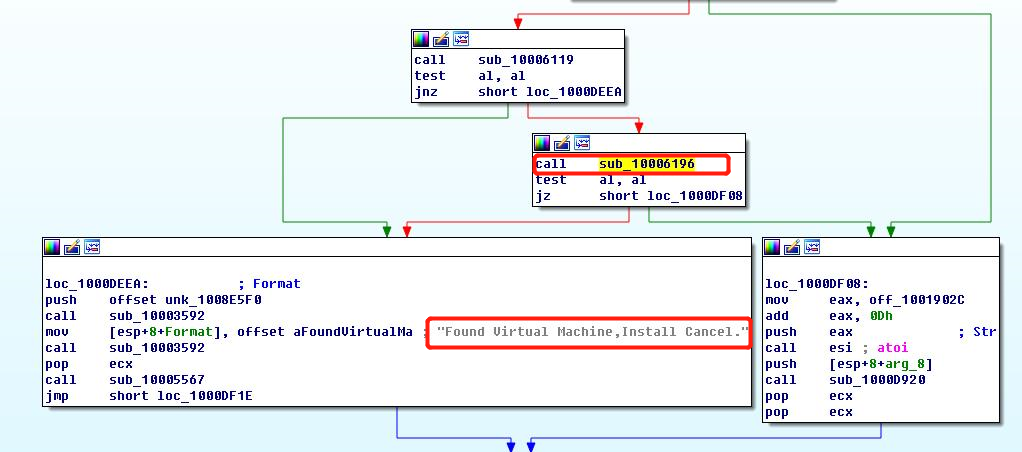


1. 搜索in指令（opcode 0xED）的使用。这个指令和一个魔术字符串VMXh用来进行VMware检测。这在这个恶意代码中被使用了吗? 使用对执行in指令函数的交叉引用，能发现进一步检测VMware的证据吗?

搜索in指令，在0x100061DB处找到，同时看到魔术字符串VMXh，可以确定这段代码是恶意代码采用的反虚拟机技术。

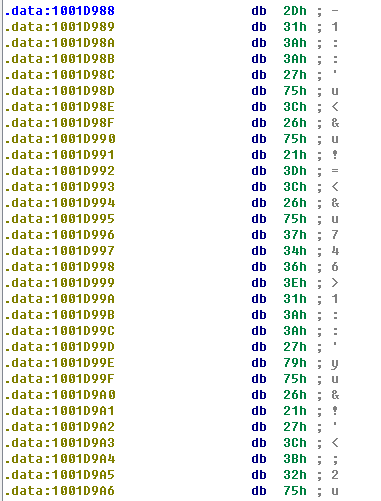


查找执行in指令函数的交叉引用，可以看到“Fonud Virtual Machine,install Cancel.”字符串。



1. 将你的光标跳转到0x1001D988处，你发现了什么?

一段字符序列，可能被加密



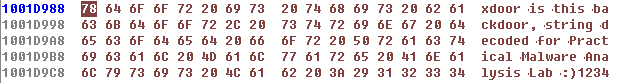
1. 如果你安装了IDA Python插件（包括IDA Pro的商业版本的插件），运行Lab05-01.py，一个本书中随恶意代码提供的IDA Pro Python脚本，（确定光标是在0x1001D988处）在你运行这个脚本后发生了什么?

该脚本应该是异或0x55执行解密运算，运行该脚本之前的字节序列变成

xdoor is this backdoor,string decoded for Practical Malware Analysis Lab : )1234

1. 将光标放在同一位置，你如何将这个数据转成一个单一的ASCII字符串?

保持光标在0x1001D988，按下A键即可，也可以在Hex View中查看结果



1. 使用一个文本编辑器打开这个脚本。它是如何工作的?

ScreenEA()获得光标当前位置，接下来它从0~0x50循环，调用Byte()获得每个函数字节值并与0x55异或，最后将其显示到IDA pro（但不会去修改原始文件）。功能上看是一段异或解密脚本。

sea = ScreenEA()  
  
for i in range(0x00,0x50):  
 b = Byte(sea+i)  
 decoded\_byte = b ^ 0x55  
 PatchByte(sea+i,decoded\_byte)

1. **编写Yara规则**

include ".\includes\IsPE.yar"

include ".\includes\SizeLimit.yar"

rule Lab5\_1

{

strings:

$string1 = "pics.praticalmalwareanalysis.com" wide ascii //DNS请求

$string2 = "This Remote Shell Session" wide ascii //远程控制

$string3 = "cmd.exe" wide ascii

$string4 = "command.exe" wide ascii

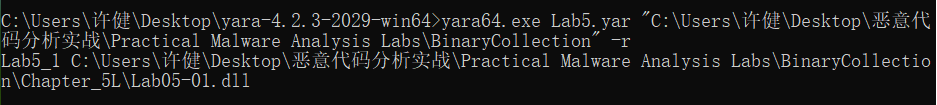
$string5 = {68 58 4D 56} //"VMXh"

$string6 = "xdoor is this backdoor" xor(0x01-0xff) //异或加密

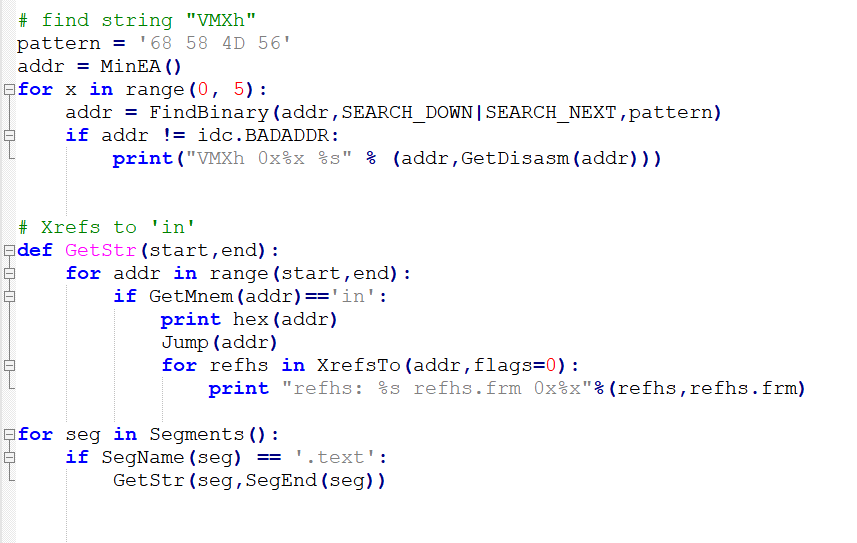
condition:

IsPE and SizeLimit and (all of ($string\*))

}



1. **编写IDAPython脚本**



编写的IDAPython脚本包括对VMXh字符串的查找和定位in指令的交叉引用

输出结果如下：

