2023-12-24

王英泰 1120210964

北京理工大学计算机学院

计算机病毒防治

实验三 IDA Pro实验报告

目录

[实验准备 3](#_Toc156228128)

[一、 实验环境和所用到的软件 3](#_Toc156228129)

[二、 基本知识 3](#_Toc156228130)

[Lab 5-1 4](#_Toc156228131)

[一、 DllMain的地址 4](#_Toc156228132)

[二、 使用导入窗口浏览gethostbyname，它在哪里导入的 4](#_Toc156228133)

[三、 有多少个函数调用gethostbyname 5](#_Toc156228134)

[四、 在0x10001757的gethostbyname的调用将产生什么DNS请求 6](#_Toc156228135)

[五、 IDA Pro识别出多少个位于0x10001656的子例程中的局部变量 7](#_Toc156228136)

[六、 IDA Pro识别出多少个位于0x10001656的子例程中的参数 8](#_Toc156228137)

[七、 使用Strings窗口去定位字符串“\cmd.exe /c” 8](#_Toc156228138)

[八、 引用字符串“\cmd.exe /c”的那段代码做了些什么 9](#_Toc156228139)

[九、 在0x100101C8的同样区域，dword\_1008E5C4似乎是全局变量去决定采取那条路径。恶意软件如何设置dword\_1008E5C4 11](#_Toc156228140)

[十、 在0x1000FF58处的子例程几百行指令中，使用了大量memcmp来进行字符串比较。如果对robotwork的字符串比较成功会发生什么 13](#_Toc156228141)

[十一、 PSLIST导出的作用 14](#_Toc156228142)

[十二、 使用图形模式来显示sub\_10004E79的交叉引用。当进入这个函数时，哪些API函数可能被调用，仅仅基于这些API函数会如何重命名这个函数 17](#_Toc156228143)

[十三、 DllMain直接调用了多个Windows API函数，当深度为2时调用了多少个 20](#_Toc156228144)

[十四、 在0x10001358处有Sleep函数调用，回看代码找出休眠时间 22](#_Toc156228145)

[十五、 在0x10001701处调用了socket函数，它的三个参数是什么 22](#_Toc156228146)

[十六、 通过MSDN查阅socket函数和IDA Pro中的命名符号常量，可以让参数的意义更明确。改变之后的参数是什么 23](#_Toc156228147)

[十七、 搜索in（操作码为0xED）指令的使用，这个指令和一个魔术字符串VMXh用来进行VMware的检测。它在恶意代码中被使用了吗？通过对in指令函数的交叉引用，可以发现进一步检测VMware的证据吗 24](#_Toc156228148)

[十八、 在0x1001D988处发现了什么 26](#_Toc156228149)

[十九、 运行Lab05-01.py脚本后发生了什么 27](#_Toc156228150)

[二十、 将该处数据转换为字符串 27](#_Toc156228151)

[二十一、 Lab05-01.py是如何工作的 28](#_Toc156228152)

[实验总结 28](#_Toc156228153)

# 实验准备

1. 实验环境和所用到的软件

* Windows 10 x86
* IDA Pro等

1. 基本知识
2. IDA Pro的操作：

* 快捷键

1. n：重命名变量或函数
2. :：添加注释
3. h/q：16进制
4. h：十进制
5. r：字符
6. b：二进制

* 插件

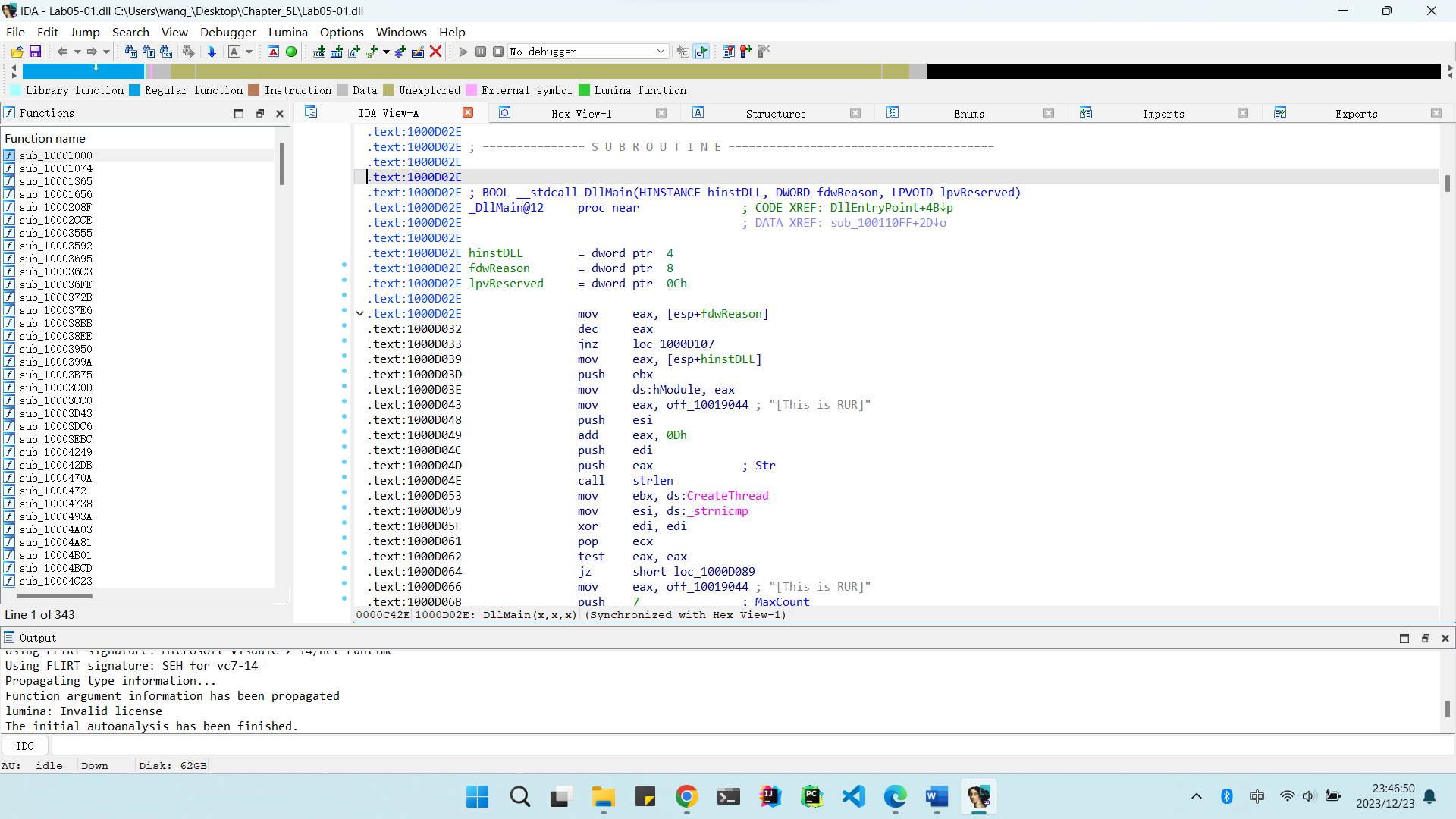
1. 使用IDA API
   1. IDC
   2. IDAPython
2. Fentanyl：给汇编文件打补丁
3. Signsrch：搜索签名
4. Ponce：符号执行

* Patch文件：Edit->Patch program
* 跳转到原文件偏移处：Jump -> Jump to File Offset
* 跳转到下一条指令：Search -> Next Code

# Lab 5-1

1. DllMain的地址

使用IDA Pro打开Lab05-01.dll，便自动跳到DllMain处，可以看到DllMain的地址为.text:1000D02E。



所有从DllEntryPoint到DllMain之间的代码是由编译器生成的，我们不需要认真分析。

1. 使用导入窗口浏览gethostbyname，它在哪里导入的

切换到Imports窗口，然后搜索gethostbyname，得到结果如下：

图形用户界面, 文本, 应用程序

描述已自动生成

双击可以看到它在idata:100163CC处导入。

图形用户界面, 文本, 应用程序

描述已自动生成

1. 有多少个函数调用gethostbyname

当选中gethostbyname选在的行后，按下“X”键，可以打开交叉引用窗口。

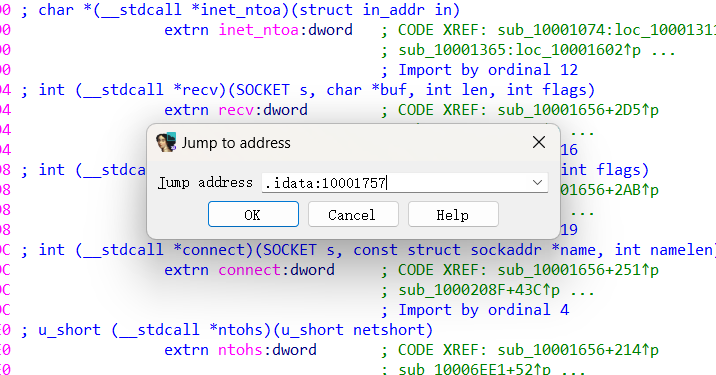
图形用户界面, 应用程序

描述已自动生成

从中可以看到gethostbyname被5个函数所调用，分别是sub\_10001074、sub\_10001365、sub\_10001656、sub\_1000208F、sub\_10002CCE。

1. 在0x10001757的gethostbyname的调用将产生什么DNS请求

按下“G”键，输入地址会跳转到指定地方。



文本

描述已自动生成

函数gethostbyname只有一个参数——指向域名的字符串指针。所以该字符串地址为off\_10019040+0Dh。

双击off\_10019040跳转到数据区定义指针地方，再次双击到字符串存储位置。

表格

低可信度描述已自动生成

加上0Dh偏移后，可以得到字符串为“pics.practicalmalwareanalysis.com”。所以该函数将查询“pics.practicalmalwareanalysis.com”的IP地址。

1. IDA Pro识别出多少个位于0x10001656的子例程中的局部变量

按下“G”键，跳转到0x10001656处。

图形用户界面, 文本, 应用程序

描述已自动生成

从上图可以看到IDA Pro识别出来的参数和局部变量。如果偏移量为负表示局部变量，这里我数出了23个。

1. IDA Pro识别出多少个位于0x10001656的子例程中的参数

同上，如果偏移量为正表示参数，这里我数出了1个。

1. 使用Strings窗口去定位字符串“\cmd.exe /c”

选择菜单栏的View->Open Subviews->Strings打开Strings窗口。搜索“\cmd.exe /c”，结果如下：

图形用户界面, 文本, 应用程序

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成

双击打开字符串所在位置，可以看到为为xdoors\_d:10095B34。

图片包含 文本

描述已自动生成

1. 引用字符串“\cmd.exe /c”的那段代码做了些什么

按下“X”键，可以发现只有一个地方引用了它。通过双击数据交叉引用，可以跳转到引用字符串的代码位置。

文本

描述已自动生成

转为图形视图，可以看到该例程调用了memcmp函数，用来比较Buf1和“cd”、“exit”、“install”、“inject”、“uptime”。

图示

低可信度描述已自动生成

从该处上方的代码可以看到某个字符串中出现了Remote Shell Session字符串。同时还发现该例程调用了recv、send函数。

文本

描述已自动生成

文本

描述已自动生成

综上所述，可以猜测该函数是一个远程shell会话。

1. 在0x100101C8的同样区域，dword\_1008E5C4似乎是全局变量去决定采取那条路径。恶意软件如何设置dword\_1008E5C4

按下“G”键，跳转到0x100101C8。

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

按下“X”键可以看到有3处引用了该全局变量，有一处修改了dword\_1008E5C4。

图形用户界面, 文本, 应用程序

描述已自动生成

跳转到修改变量的地方，我们可以看到在调用了sub\_10003659后将其返回值赋给了dword\_1008E5C4。

文本

描述已自动生成

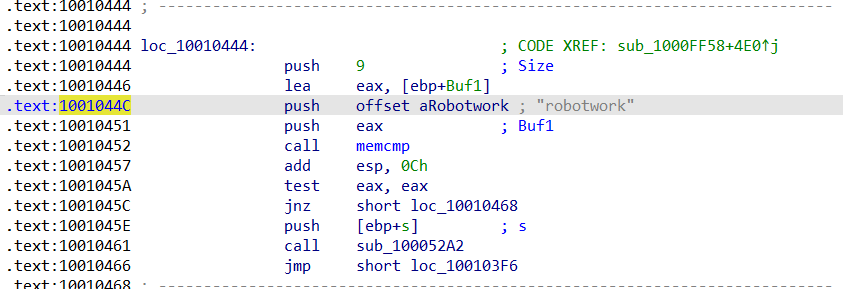
查看sub\_1008E5C4函数，我们可以发现该函数调用了GetVersionEx来获取当前机器上的OS版本，如果当前机器的OS版本是VER\_PLATFORM\_WIN32\_NT，即Windows 2000及以上，则AL被设置为1，否则返回0。

图形用户界面, 文本, 应用程序

描述已自动生成

1. 在0x1000FF58处的子例程几百行指令中，使用了大量memcmp来进行字符串比较。如果对robotwork的字符串比较成功会发生什么

按下“G”键，跳转到0x1000FF58处。在.text:1001044C处，我们找到了字符串“robotwork”。



可以发现，如果比较成功，则会调用sub\_1000525A函数。查看该函数，可以发现它查询在HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ WorkTime 中的WorkTimes值，然后调用了函数sub\_100038EE。

图形用户界面, 文本, 应用程序

描述已自动生成

查看sub\_100038EE函数，可以发现它调用了send函数将信息发送。

图形用户界面

中度可信度描述已自动生成

1. PSLIST导出的作用

在Exports窗口中找到PSLIST，然后双击转到代码处。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

该函数调用了sub\_100036C3函数，查看该函数，该函数的作用是判断该OS的版本是否在Windows Vista及以上。根据参数Str是否为空调用不同的函数sub\_10006518和sub\_1000664C。

图形用户界面, 文本, 应用程序

描述已自动生成

函数sub\_10006518和sub\_1000664C都调用了CreateToolhelp32Snapshot函数，用来获取进程列表。所以该函数的作用可能是将系统进程信息发送给远程。

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

1. 使用图形模式来显示sub\_10004E79的交叉引用。当进入这个函数时，哪些API函数可能被调用，仅仅基于这些API函数会如何重命名这个函数

在左侧Functions窗口中找到sub\_10004E79，选中函数然后选择View->Graphs -> Xrefs From从而打开交叉引用的图形模式。

表格

描述已自动生成

图示

描述已自动生成

从图中可以看出，该函数调用了GetSystemDefaultLangId、send等函数，所以这个函数的主要功能是获取当前系统的默认语言并发送，可以将这个函数重命名为GetLangIdAndSend。

在函数窗口右击sub\_10004E79选择Edit Function来重命名函数。

图形用户界面, 应用程序

描述已自动生成

1. DllMain直接调用了多个Windows API函数，当深度为2时调用了多少个

在Functions窗口找到DllMain函数，然后选择菜单栏的View->Graphs->User xrefs chart。

图形用户界面, 应用程序

描述已自动生成

起始地址和结束地址都写DllMain的起始地址，然后选取深度为1绘图。

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 图示

描述已自动生成

将深度改为2，得到如下图：

图形用户界面, 文本, 应用程序

描述已自动生成

1. 在0x10001358处有Sleep函数调用，回看代码找出休眠时间

按下“G”键输入跳转地址，跳转到0x10001358处，我们发现Sleep函数的参数放在eax中。

图形用户界面, 应用程序

描述已自动生成

分析以上代码，我们可以得出首先将字符串地址加上偏移转换为数字，然后乘3E8h。所以休眠时间为30秒。

1. 在0x10001701处调用了socket函数，它的三个参数是什么

按下“G”键输入地址跳转到0x10001701处。

图片包含 文本

描述已自动生成

从上面可以看出它的三个参数是2、1、6。

1. 通过MSDN查阅socket函数和IDA Pro中的命名符号常量，可以让参数的意义更明确。改变之后的参数是什么

通过查阅[MSDN文档](https://learn.microsoft.com/zh-cn/windows/win32/api/winsock2/nf-winsock2-socket)可以得到这些参数的含义。

图形用户界面, 文本, 应用程序

描述已自动生成

从该文档中可知，第一个参数为af，表示地址系列规范，2对应的常量为AF\_INET，表示Internet 协议版本 4 (IPv4) 地址系列；第二个参数为type，表示新套接字的类型规范，1代表的常量为SOCK\_STREAM，表示一种套接字类型，它通过 OOB 数据传输机制提供排序的、可靠的双向、基于连接的字节流。 此套接字类型对 Internet 地址系列 (AF\_INET 或AF\_INET6) 使用传输控制协议 (TCP)；第三个参数为protocol，表示要使用的协议，6代表的常量为IPPROTO\_TCP表示传输控协议TCP。

选中数字右键选择Symbolic constant->Use standard sysmbolic constant，将其转换为正确的常量名。

图形用户界面, 应用程序

描述已自动生成

转换后的结果如下：

文本

中度可信度描述已自动生成

1. 搜索in（操作码为0xED）指令的使用，这个指令和一个魔术字符串VMXh用来进行VMware的检测。它在恶意代码中被使用了吗？通过对in指令函数的交叉引用，可以发现进一步检测VMware的证据吗

选择菜单栏中的Search->Sequence of bytes，输入in指令的操作码ED，并勾选Find all occurrences。找到唯一的in语句。

图形用户界面, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

按下“R”键将地址转换为字符串。

图形用户界面

描述已自动生成

从这里我们就可以看出这个恶意软件使用了该方法来检测VMware。

找到函数头，按下“X”键打开交叉引用的窗口。

图形用户界面, 文本, 应用程序

描述已自动生成

随便选择一个函数，我们发现在调用了该函数后，会进行判断，如果为虚拟机，就会产生更为详细的信息“Found Virtual Machine,Install Cancel.”

图形用户界面, 应用程序

描述已自动生成

1. 在0x1001D988处发现了什么

按下“G”键输入地址进行跳转，我们发现了没有意义的数据。

表格

中度可信度描述已自动生成

1. 运行Lab05-01.py脚本后发生了什么

选中当前行，然后选择菜单栏中的File->Script File，并选择Python脚本。这里我没有IDA Python插件，所以无法显示效果。

根据答案发现这里的字符串改变。

1. 将该处数据转换为字符串

选中字符串开头位置，按下“A”键，将其转换为字符串。

表格

低可信度描述已自动生成

1. Lab05-01.py是如何工作的

查看Lab05-01.py，我们发现该脚本的原理如下：

文本

描述已自动生成

sea = ScreenEA()获取光标当前位置，接下来对0x0~0x50之间的字符串进行处理，通过Byte函数获取该处的字符，将其异或0x55，然后再写入文件。

# 实验总结

本章介绍了IDA Pro的使用方法。IDA Pro可以说是这门课程最大的收获之一。在上这门课之前，我还不知道有这么一个软件可以用来方便地分析二进制文件并进行反汇编。在刚接触IDA时，它的英文界面和繁多的窗口让我不知所措。但是，经过这次实验和以后的几次实验后，我对IDA Pro的操作已经很熟练了。