**恶意代码分析与防治技术实验报告**

**Lab5**

1. **实验环境**

Windows7，VMWARE，Windows11

1. **实验工具**

STRINGS, IDAPro,PEVIEW，EXEInfo，YARA

1. **实验内容**

Lab 5

1. What is the address of DllMain?

.text:1000D02E

分析过程：

使用 IDA 打开 Lab05-01.dll，如图，即可得到 DllMain 所在地址，即.text 节的 0x1000D02E 处。

2. Use the Imports window to browse to gethostbyname. Where is the import located?

.idata:100163CC

分析过程：

打开 Imports 窗口，搜索该函数，双击进入，即可得到地址为 idata 节的 0x100163CC 处。

3. How many functions call gethostbyname?

5 个函数，调用了 9 次。

分析过程：

右击刚刚找到的 gethostbyname 函数，进入 Jump tp xref to oprand 选项，其中类型 p 是指被

调用的引用，类型 r 是指被读取的引用，CPU 是先读取再调用的，且仔细观察可以看到，这

18 项中有很多一样的地址（即是同一个函数调用的），所以这里只有五个函数调用了

gethostbyname，共调用了九次。

4. Focusing on the call to gethostbyname located at 0x10001757, can you figure out which DNS

request will be made?

对 pics.practicalmalwareanalysis.com 进行 DNS 解析。

分析过程：

跳转到该地址，分析该函数调用附近的代码：首先将 off\_10019040 处的值赋给 eax；由于

off\_10019040 是一个指针，因此下一步将 eax+0Dh，即是将指针后移 13 位，存入 eax；将

eax 压栈，作为 gethostbyname 的参数，调用函数。

下面我们跳转到 off\_10019040 处，查看存储的是什么数据。

可以看到存储的字符串是“[This is RDO]pics.praticalmalwareanalysis.com”，后移 13 位即

“pics.practicalmalwareanalysis.com”，因此，会对 pics.practicalmalwareanalysis.com 进行 DNS

解析。

5. How many local variables has IDA Pro recognized for the subroutine at 0x10001656?

24。

分析过程：

跳转到该地址，可以发现共 24 个。

6. How many parameters has IDA Pro recognized for the subroutine at 0x10001656？

1 个。

7. Use the Strings window to locate the string \cmd.exe /c in the disassembly. Where is it located?

Xdoors\_d:10095B34

8. What is happening in the area of code that references \cmd.exe /c?

猜测发生了一个 shell 远程会话函数。

分析过程：

右击找到该程序被引用的地方，发现只有一处，双击进去，发现该字符串被压栈。

切换到图形模式，可以看到如下一系列的 memcmp 函数被调用，用于比较“quit”、“cd”、

“exit”、“uptime”等字符串，都是关于系统信息和系统操作的。

且在字符串“\cmd.exe /c”被压栈的指令附近，还发现了如下字符串，结合 recv 和 send 调

用，可以猜测是一个 shell 远程会话函数。

9. In the same area, at 0x100101C8, it looks like dword\_1008E5C4 is a global variable that helps

decide which path to take. How does the malware set dword\_1008E5C4? (Hint: Use

dword\_1008E5C4’s cross-references.)

通过调用函数比较“2”与当前操作系统版本的信息，通过 setz 指令将该函数返回值设置为

1，又把返回值 1 赋给 dword\_1008E5C4。因此，该全局变量实际上存的是操作系统的版本

信息。

分析过程：

跳转到 0x100101C8 处，双击该全局变量，来到.data:0x1008E5C4 处，查看它的交叉引用，

可以发现被调用了三次。其中只有一次是 mov 操作，修改了 dword\_1008E5C4 的值，点进去

查看附近的代码。

它是将 sub\_10003695 的返回值（保存在 eax 中）赋给了 dword\_1008E5C4，接下来双击

sub\_10003695，看看这个函数的返回值是什么。

如 图 ， 在 该 函 数 中 调 用 了 GetVersionExA 来 获 取 当 前 操 作 系 统 版 本 的 信 息 ， 将

[ebp+VersionInformation.dwPlatformId] （ 即 dwPlatformId） 与 2 （ 2 代 表 VER\_PLATFORM\_ WIN32\_NT）进行比较，通过比较结果来设置 AL 寄存器，从而得到返回值。

根本当前操作系统的版本号（Windowsxp）以及 setz 的规则，我们可以知道，该全局变量被

设置成了 1。

10. A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to

compare strings. What happens if the string comparison to robotwork is successful (when memcmp

returns 0)?

会查询注册表中 HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTime 和

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTimes 的键值，并通过 socket 传

递。

分析过程：

跳转到 0x1000FF58 处，该函数有几百行，调用了一系列的 memcmp 函数。其中在 0x10010452

处发生了与 robotwork 的比较，分析这个过程。其中 test eax,eax 检测 memcmp 的返回值是

否为 0，若是 0（即比较的字符串就是“robotwork”），则下一句 jnz 指令就不会跳转，而

将调用后两行的 sub\_100052A2 处的函数，参数是[ebp+s]。

我们双击进入 sub\_100052A2 处，可以发现它查询了注册表中

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTime 和

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTimes 的键值，并将消息返回给了

[ebp+s]处的 socket。

11. What does the export PSLIST do?

该导出项可以根据操作系统的版本，获得进程列表或指定进程的相关信息，并通过 socket

发送。

分析过程：

查看 Exports 表，双击 PSLIST 进入到该导出项代码的起始处，发现这个函数会根据

sub\_100036C3 处的返回值，选择两个不同的路径，而这两条路径都会使用函数

CreateToolhelp32Snapshot 获得进程列表或指定进程的信息，并通过 socket 发送。

进入 sub\_100036C3，发现该函数是检查操作系统的版本的。

12. Use the graph mode to graph the cross-references from sub\_10004E79. Which API functions

could be called by entering this function? Based on the API functions alone, what could you rename

this function?

调用的 API 函数有 GetSystemDefaultLangID、sprintf、send。

可以为它重命名：send\_SystemDefaultLangID。

分析过程：

跳转到 sub\_10004E79 处，通过菜单的 View-Graphs->User Xrefs Chart 得到下图中的交叉引用

图。可以看到，该函数调用了很多函数，其中，通过 GetSystemDefaultLangID 和 send 可以推

测 该 函 数 可 能 通 过 socket 发 送 语 言 标 志 。 根 据 功 能 ， 可 以 给 这 个 函 数 重 命 名 为

send\_SystemDefaultLangID。

13. How many Windows API functions does DllMain call directly? How many at a depth of 2?

DllMain 直接调用的 API 函数有：strncpy、\_strnicmp、CreateThread 和 strlen；递归深度为 2

时调用了很多 API 函数，包括 Sleep、WinExec、gethostbyname 及一些网络函数。

分析过程：

搜索 DllMain 找到并跳转到这个函数的位置，按照与上面同样的方法打开交叉引用图：

发现极其复杂，于是要更改一些查询条件。将递归深度设置为 1（只显示被它直接调用的函

数），其中 API 函数有：strncpy、\_strnicmp、CreateThread 和 strlen。

再次修改，将递归深度设置为 2，又得到了一张很复杂的图，包括 Sleep、gethostbyname、

WinExec 等函数。

14. At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the

number of milliseconds to sleep). Looking backward through the code, how long will the program

sleep if this code executes?

30s。

分析过程：

跳转到 0x10001358 处，可以看到这个函数只有一个参数：eax 处的值，而往上看，eax 处存

的是函数 atoi 的返回值乘上 3E8h（即十进制下的 1000）的结果。再继续往上看，先将

off\_10019020（点进去可以发现这是个字符串“[This is CTI]30”）赋给 eax，再加上 0xD，即

向后移动 13 位，得到新的字符串“30”，作为参数传给函数 atoi，变成数字 30，再乘上 1000，

就成了 30000ms=30s。于是这个程序将休眠 30s。

15. At 0x10001701 is a call to socket. What are the three parameters?

三个参数从右往左依次是 6、1、2。

分析过程：

跳转到 0x10001701 处，该函数的 3 个参数从右往左依次是 6、1、2。

16. Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can

you make the parameters more meaningful? What are the parameters after you apply changes?

there other ways to run this program?

6 指的是 IPPROTO\_TCP、1 指的是 SOCK\_STREAM、2 指的是 AF\_INET。

分析过程：

如上图，三个参数后面的注释也告诉了我们这些参数与 socket 描述中的符号常量相关。右

击每个数，选择 Use standard symbolic constant，弹出的对话框列举了 IDA Pro 为这个特定值

找到的所有的对应常量。在这里，6 指的是 IPPROTO\_TCP、1 指的是 SOCK\_STREAM、2 指的

是 AF\_INET。替换后如下：

17. Search for usage of the in instruction (opcode 0xED). This instruction is used with a magic string

VMXh to perform VMware detection. Is that in use in this malware? Using the cross-references to

the function that executes the in instruction, is there further evidence of VMware detection?

在这个恶意代码中也使用了，检查这个函数的交叉引用，还可以在调用函数中发现字符串

“Found Virtual Machine”。

分析过程：

通过 Search->Text 来搜索 in 指令，出现了很多条结果，其中只在 100061DB 处有一条 in 指令，

点进去查看该指令。

将 564D5868h 赋给 eax，右击该值，可以发现它在 ASCII 码下的值是“VMXh”。检查这个函

数的交叉引用，还可以在调用函数中发现字符串“Found Virtual Machine”。

18. Jump your cursor to 0x1001D988. What do you find?

可以看到一些看起来随机的字符串。

分析过程：

跳转到 0x1001D988 处，可以看到一系列的无规律不可读字符串，即乱码。

19. If you have the IDA Python plug-in installed (included with the com- mercial version of IDA

Pro), run Lab05-01.py, an IDA Pro Python script provided with the malware for this book. (Make

sure the cursor is at 0x1001D988.) What happens after you run the script?

通过 File->Script File，选择 Lab05-01.py 运行脚本后，可以看到 0x1001D988 处的数据变得可

读。

20. With the cursor in the same location, how do you turn this data into a single ASCII string?

保持光标在 0x1001D988 处，按下 A 键即可转换，转换成的 ASCII 字符串是：

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

21. Open the script with a text editor. How does it work?

结合 Python 脚本，我们知道该脚本首先获取光标的当前位置，作为解码数据的偏移值。接

下来，从 0x00~0x50 循环，将读到的字节与 0x55 进行异或，并将结果写回 IDA Pro 的显示

中，而不修改原文件。

1. **实验心得**

**本次实验，我熟悉了对恶意代码分析工具有了更深入的理解。同时得到了充分的锻炼，并开始对本门课程的实验开始得心应手，做实验的速度越来越快了。**