恶意代码分析实战Lab0501

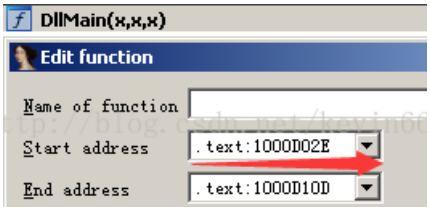
原创

Flying_Fatty 于 2018-02-08 07:28:26 发布 237 收藏
分类专栏： 恶意代码分析实战 文章标签： 恶意代码分析实战课后习题

本文链接：https://blog.csdn.net/kevin66654/article/details/79271536


恶意代码分析实战 专栏收录该内容
43 篇文章 2 订阅
订阅专栏

问题1：DllMain的地址

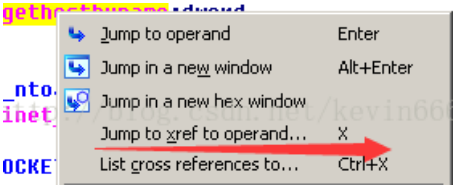


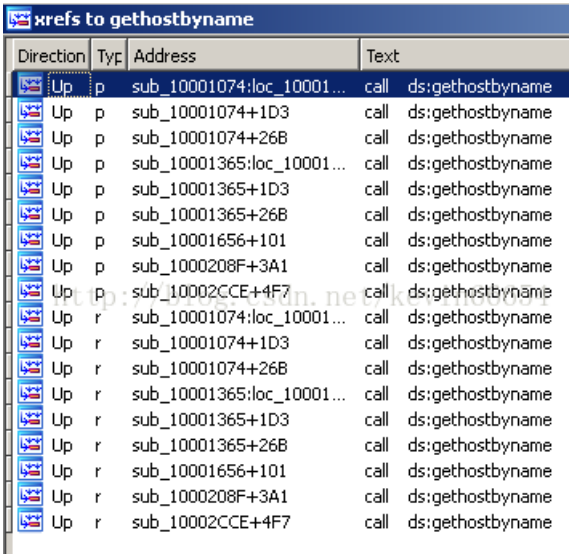
地址是1000D02E

问题2：使用Imports窗口浏览到gethostbyname，导入函数定位到什么地址

地址是100163CC

问题3：有多少函数调用了gethostbyname





问题4：在位于0x10001757处对于gethostbyname的调用，触发的是哪个DNS请求

如上图所示，1757的调用时图上的第六个，所以我们来分析1656这个函数即可

```
if ( memcmp(off_10019040[0] + 13, asc_10093540, 0x10u) && !dword_1008E5CC )
{
  v8 = gethostbyname((const char *)off_10019040[0] + 13);
  v9 = v8;
  if ( v8 )
  {
    memcpy(&Dst, *(const void **)v8->h_addr_list, v8->h_length);
    v45 = v9->h_addrtype;
    v10 = inet_ntoa(Dst);
    strncpy(cp, v10, 0x10u);
    strncpy(::Str, (const char *)off_10019038[0] + 13, 5u);
    WinExec(CmdLine, 0);
  }
```

```
off_10019040    dd offset aThisIsRdoPics_
                                        ; DATA XREF: sub_10001656:loc_10001722↑r
                aThisIsRdoPics_ db '[This is RDO]pics.praticalmalwareanalysis.com',0
```

在函数中找到gethostbyname的调用，前后观察其重要的常量字符串

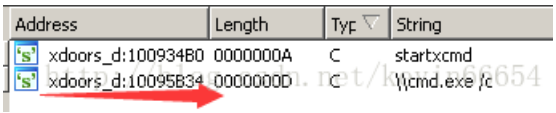
问题5、6：IDA识别了1656函数的多少个局部变量和参数

这个问题其实没有太大意义，知道怎么操作就好。

当前版本的IDA，识别的是一个参数lpThreadParameter，以及62个参数

问题7：定位\cmd.exe /c.

Shift + F12，查看Strings



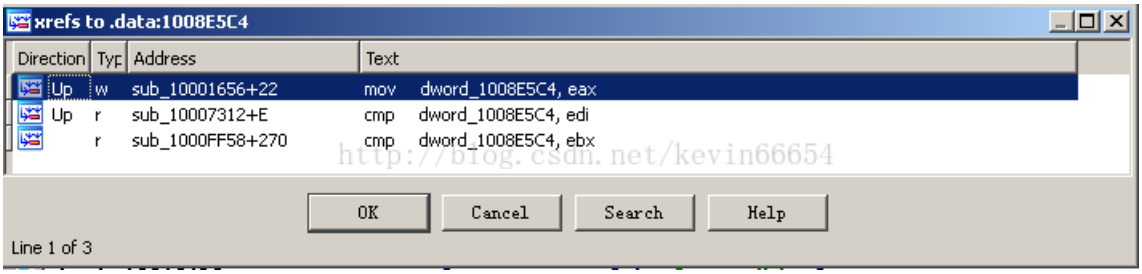
问题8：在\cmd.exe /c的附近发生了什么

```
.text:100101C2                call    ds:GetSystemDirectoryA
.text:100101C8                cmp     dword_1008E5C4, ebx
.text:100101CE                jz      short loc_100101D7
.text:100101D0                push    offset aCmd_exeC ; "\\cmd.exe /c "
.text:100101D5                jmp     short loc_100101DC
```

发现到了GetSystemDirectory的API函数调用

问题9：观察dword_1008E5C4的交叉引用

查看全局变量的交叉引用如图



根据引用来看，恶意代码是使用eax，edi，ebx这些寄存器来赋值该全局变量

```
.text:10001656                sub     esp, 678h
.text:1000165C                push    ebx
.text:1000165D                push    ebp
.text:1000165E                push    esi
.text:1000165F                push    edi
.text:10001660                call    sub_10001000
.text:10001665                test    eax, eax
.text:10001667                jnz     short loc_100016BC
.text:10001669                xor     ebx, ebx
.text:1000166B                mov     [esp+688h+var_674], ebx
.text:1000166F                mov     [esp+688h+hModule], ebx
.text:10001673                call    sub_10003695
.text:10001678                mov     dword_1008E5C4, eax
.text:1000167D                call    sub_100036C3
```

eax是多少？在1665的test eax,eax时，当eax为0时，jnz不成立，会顺序往下走。即当eax=0时，全局变量1008E5C4为0，会顺序执行下面的Sleep和WSAStartup函数

```
.text:10007312                push    ebp
.text:10007313                mov     ebp, esp
.text:10007315                sub     esp, 208h
.text:1000731B                push    ebx
.text:1000731C                push    esi
.text:1000731D                push    edi
.text:1000731E                xor     edi, edi
.text:10007320                cmp     dword_1008E5C4, edi
.text:10007326                jz      loc_100073D2
.text:1000732C                call    ds:GetCurrentThreadId
.text:10007332                push    eax             ; dwThreadId
.text:10007333                call    ds:GetThreadDesktop
.text:10007339                push    400001CFh       ; dwDesiredAccess
.text:1000733E                push    edi             ; fInherit
.text:1000733F                push    edi             ; dwFlags
.text:10007340                mov     esi, eax
.text:10007342                call    ds:OpenInputDesktop
```

xor edi,edi即赋值edi为0.

当edi为0时，cmp之后有个jz，所以是当全局变量不为0时，执行GetCurrentThreadId与GetThreadDesktop等函数

```
.text:100101A5                 push    400h            ; uSize
.text:100101AA                 mov     [ebp+StartupInfo.hStdError], eax
.text:100101AD                 mov     [ebp+StartupInfo.hStdOutput], eax
.text:100101B0                 lea     eax, [ebp+CommandLine]
.text:100101B6                 mov     [ebp+StartupInfo.wShowWindow], bx
.text:100101BA                 push    eax             ; lpBuffer
.text:100101BB                 mov     [ebp+StartupInfo.dwFlags], 101h
.text:100101C2                 call    ds:GetSystemDirectoryA
.text:100101C8                 cmp     dword_1008E5C4, ebx
.text:100101CE                 jz      short loc_100101D7
.text:100101D0                 push    offset aCmd_exeC ; "\\cmd.exe /c "
.text:100101D5                 jmp     short loc_100101DC
```

这里就看不出啥区别了，待会儿看完writeup来补思路

问题10：观察0x1000FF58处的多处memcmp.robotwork如果成功匹配（返回值为0），会发生什么

这里看到多个memcmp，且是多种字符串的匹配，很像恶意代码控制台的选择menu

```
xdoors_d:10095B04 aEnmagic        db 'enmagic',0          ; DATA XREF: sub_1000FF58+429To
xdoors_d:10095B0C aCd             db 'cd',0               ; DATA XREF: sub_1000FF58+3AAto
xdoors_d:10095B0F                 align 10h
xdoors_d:10095B10 aExit           db 'exit',0             ; DATA XREF: sub_1000FF58+38Dto
xdoors_d:10095B15                 align 4
xdoors_d:10095B18 aQuit           db 'quit',0             ; DATA XREF: sub_1000FF58+36Fto
xdoors_d:10095B1D                 align 10h
xdoors_d:10095B20 ; char aCommand_exeC[]
xdoors_d:10095B20 aCommand_exeC   db '\command.exe /c ',0 ; DATA XREF: sub_1000FF58:loc_100101D7to
xdoors_d:10095B31                 align 4
xdoors_d:10095B34 aCmd_exeC       db '\cmd.exe /c ',0     ; DATA XREF: sub_1000FF58+278to
xdoors_d:10095B41                 align 4
xdoors_d:10095B44 ; char aHiMasterDDDDDD[]
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
xdoors_d:10095B44                                         ; DATA XREF: sub_1000FF58+145to
xdoors_d:10095B44                 db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
xdoors_d:10095B44                 db 0Dh,0Ah
xdoors_d:10095B44                 db 'Machine UpTime  [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Secon'
xdoors_d:10095B44                 db 'ds]',0Dh,0Ah
xdoors_d:10095B44                 db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095B44                 db 'nds]',0Dh,0Ah
xdoors_d:10095B44                 db 0Dh,0Ah
xdoors_d:10095B44                 db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095B44                 db 0Dh,0Ah,0
```

比如说登陆之后的提示符，cd、exit等执行，很像恶意代码控制台的选择menu

Address	Length	Typ	String
xdoors_d:10094094	0000001E	C	\r\n\r\n[Robot_WorkTimes:] %d\r\n\r\n
xdoors_d:100940...	0000001E	C	\r\n\r\n[Robot_WorkTime :] %d\r\n\r\n
xdoors_d:10095A...	0000000A	C	robotwork

利用Strings功能，找到robotwork

```
else if ( !memcmp(&Dst, aUptime, 6u) )
{
  sub_10004DCA(s);
}
else if ( !memcmp(&Dst, aLanguage, 8u) )
{
  sub_10004E79(s);
}
else if ( !memcmp(&Dst, aRobotwork, 9u) )
{
  sub_100052A2(s);
}
else if ( !memcmp(&Dst, aMbase, 5u) )
{
  sub_10004EE0(s);
}
```

说明一个字符串的成功匹配，会调用到一个一一对应的函数，robotwork对应的是100052A2

```
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, aSoftwareMicros, 0, 0xF003Fu, &phkResult) )
{
  result = RegCloseKey(phkResult);
}
else
{
  if ( !RegQueryValueExA(phkResult, aWorktime, 0, &Type, &Data, &cbData) )
  {
    v2 = atoi((const char *)&Data);
    sprintf(&Dest, aRobot_worktime, v2);
    v3 = strlen(&Dest);
    sub_100038EE(s, (int)&Dest, v3);
  }
  memset(&Data, 0, 0x200u);
  if ( !RegQueryValueExA(phkResult, aWorktimes, 0, &Type, &Data, &cbData) )
  {
    v4 = atoi((const char *)&Data);
    sprintf(&Dest, aRobot_workti_0, v4);
    v5 = strlen(&Dest);
    sub_100038EE(s, (int)&Dest, v5);
  }
  result = RegCloseKey(phkResult);
}
return result;
```


可见，其功能是完成注册表的新建

```
xdoors_d:10093A50 ; CHAR aSoftwareMicros[]
xdoors_d:10093A50 aSoftwareMicros db 'SOFTWARE\Microsoft\Windows\CurrentVersion',0
```

问题11：PSLIST导出函数

可见，其功能是完成注册表的新建

```
result = sub_100036C3();
if ( result )
{
  if ( strlen(Str) )
    result = sub_1000664C(a1, 0, Str);
  else
    result = sub_10006518();
}
dword_1008E5BC = 0;
return result;
```


分析其中调用的几个函数

36C3：调用API函数GetVersionEx，在一个OSVERSIONINFO结构中载入与平台和操作系统有关的版本信息

664C：

```
if ( Process32First(hSnapshot, &pe) )
{
  do
  {
    v5 = strlen(Str);
    if ( !strnicmp(pe.szExeFile, Str, v5) )
    {
      v6 = OpenProcess(0x410u, 0, pe.th32ProcessID);
      EnumProcessModules(v6, &hModule, 0x1000u, &cbNeeded);
      memset(&Dst, 0, 0x104u);
      GetModuleFileNameExA(v6, hModule, &Dst, 0x104u);
      sprintf(&Dest, a16d20sD, pe.th32ProcessID, pe.szExeFile, pe.cntThreads);
      sub_100038BB(s, &Dest);
      sprintf(&Dest, aS_1, &Dst);
      sub_100038BB(s, &Dest);
      if ( dword_1008E5BC )
        sub_1000620C(a16d20sDS, pe.th32ProcessID);
      CloseHandle(v6);
    }
  }
  while ( Process32Next(hSnapshot, &pe) );
}
else
{
  v7 = GetLastError();
  sprintf(&Dest, aProcess32first, v7);
  sub_100038BB(s, &Dest);
}
```


在进程列表中搜索所有进程，其中Process32First是获取当前系统中运行的第一个进程，while中的Process32Next是获取下一个进程，形成的链表结构确保能够不重复不遗漏的遍历所有进程。从匹配进程名称来看，应该是恶意代码对于某种类型的系统进程实现了某种监听或者注入

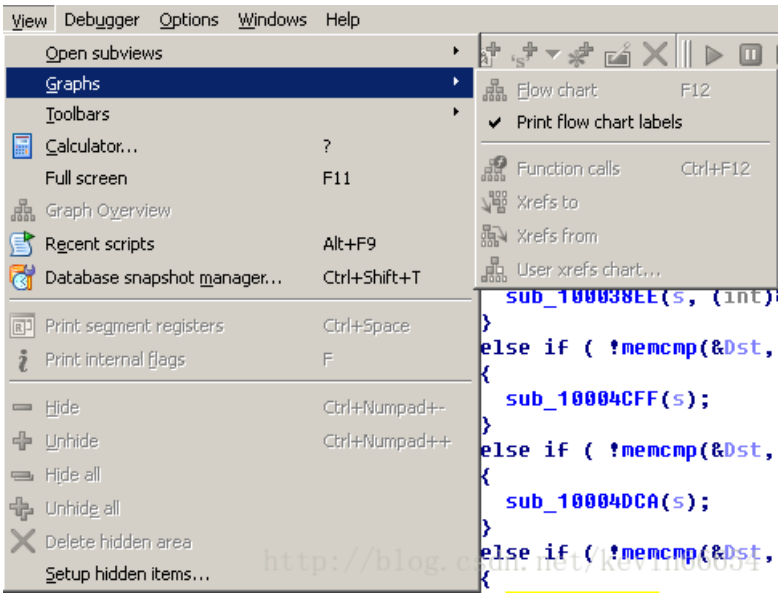
6518：

```
for ( i = Process32First(hSnapshot, &pe); i; i = Process32Next(hSnapshot, &pe) )
{
  v1 = OpenProcess(0x410u, 0, pe.th32ProcessID);
  EnumProcessModules(v1, &hModule, 0x1000u, &cbNeeded);
  memset(&Dst, 0, 0x104u);
  GetModuleFileNameExA(v1, hModule, &Dst, 0x400u);
  if ( dword_1008E5BC )
    sub_1000620C(a16d20sDS, pe.th32ProcessID);
  CloseHandle(v1);
}
```


和上面的664C框架结构类似

猜想：该导出函数实现的是对进程中的含有特殊字符串进程的监听或者是注入

问题12：分析函数10004E79



不知道为啥不能调用出函数的交叉引用图来。4E79调用了GetSystemDefaultLangID。查看的是系统的默认语言环境。然后是被1656调用

截进去看一眼，果然就是之前分析的，多个特征字符串匹配，多个函数调用的地方。重命名为LanguageCheck即可（简洁易读）

问题13：DllMain直接调用了多少个Windows API？

第一眼看到的是strnicmp，CreateThread，strncpy这几个

深度为2的需要进去子函数分析

问题14：0x10001358的Sleep时长

```
v17 = atoi((const char *)off_10019020[0] + 13);
Sleep(1000 * v17);
```

这里睡眠的时间是v17的数值，单位是秒。30 + 13 = 43秒

问题15，16：1701处socket调用的参数

```
.text:100016FB                push    6               ; protocol
.text:100016FD                push    1               ; type
.text:100016FF                push    2               ; af
.text:10001701                call    ds:socket
```

这里就是标准socket的使用。IDA已经有注释了，protocol，type，af

还是那句话，不加壳不加花的代码是很明显很好分析的，因为作者在写代码的时候是有逻辑的，找到那条主线结合正常的猜想，大致功能基本不会差太多的

根据网上的资料，protocol有如下常见的值：IPPROTO_TCP(6)。当内核向此raw socket交付数据包的时候，是包括整个IP头的，并且已经是重组好的IP包。protocol是IPPROTO_RAW(255)，这时候，这个socket只能用来发送IP包，而不能接收任何的数据。protocol为0（IPPROTO_IP)的raw socket。用于接收任何的IP数据包。

type如下：SOCK_STREAM = 1 （stream connection socket），SOCK_DGRAM = 2， SOCK_RAW = 3，SOCK_RDM = 4，SOCK_SEQPACKET = 5，SOCK_DCCP = 6，SOCK_PACKET = 10

family如下：AF_INET = 2

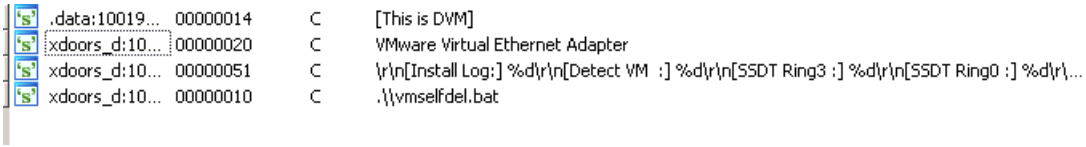
函数调用形式为socket（int family，int type，int protocol）

问题17：搜索in指令的使用。程序是如何进行VMware检测的？

AF_INET = 2

aa

在strings中搜索VM




```
46  sub_100038EE(s, (int)&Dest, v6);
47  sprintf(&Dest, aHost_name);
48  v7 = strlen((const char *)off_10019048[0] + 13);
49  strncat(&Dest, (const char *)off_10019048[0] + 13, v7);
50  v8 = strlen(&Dest);
51  sub_100038EE(s, (int)&Dest, v8);
52  sprintf(&Dest, aCurl);
53  v9 = strlen((const char *)off_10019044[0] + 13);
54  strncat(&Dest, (const char *)off_10019044[0] + 13, v9);
55  v10 = strlen(&Dest);
56  sub_100038EE(s, (int)&Dest, v10);
57  sprintf(&Dest, aDomain_name);
58  v11 = strlen((const char *)off_10019040[0] + 13);
59  strncat(&Dest, (const char *)off_10019040[0] + 13, v11);
60  v12 = strlen(&Dest);
61  sub_100038EE(s, (int)&Dest, v12);
62  v13 = atoi((const char *)off_10019020[0] + 13);
63  v14 = atoi((const char *)off_10019024[0] + 13);
64  sprintf(&Dest, aHostConnectTyp, dword_1008E5CC, v14, v13);
65  v15 = strlen(&Dest);
66  sub_100038EE(s, (int)&Dest, v15);
67  v16 = atoi((const char *)off_10019030[0] + 13);
68  v17 = atoi((const char *)off_1001902C[0] + 13);
69  v18 = atoi((const char *)off_10019034[0] + 13);
70  v19 = atoi((const char *)off_10019028[0] + 13);
71  sprintf(&Dest, aInstallLogDDet, v19, v18, v17, v16);
72  v20 = strlen(&Dest);
73  return sub_100038EE(s, (int)&Dest, v20);
74}
000044ED sub_10004EE0:61
```

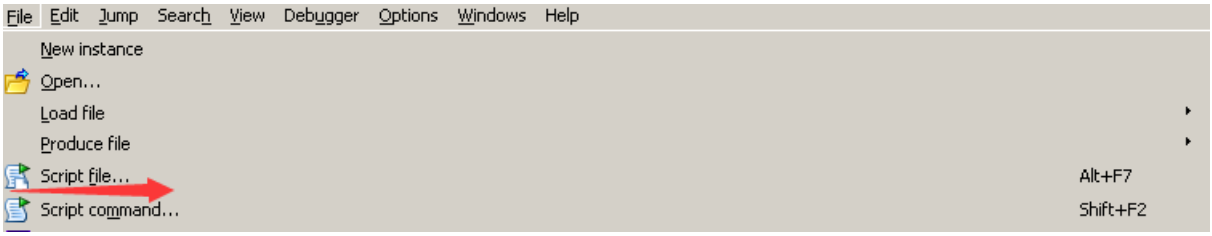
4EE0这个函数，在本地找寻机器的各种端口连接信息，其中aInstallLogDDet中，在检测VM

问题18、19、20：在0x1001D988处运行IDA-py，并且修改显示格式

在汇编处跳转到地址0x1001D988

```
.data:1001D984                  db    0
.data:1001D985                  db    0
.data:1001D986                  db    0
.data:1001D987                  db    0
.data:1001D988                  db  2Dh ; -
.data:1001D989                  db  31h ; 1
.data:1001D98A                  db  3Ah ; :
.data:1001D98B                  db  3Ah ; :
.data:1001D98C                  db  27h ; '
.data:1001D98D                  db  75h ; u
.data:1001D98E                  db  3Ch ; <
.data:1001D98F                  db  26h ; &
.data:1001D990                  db  75h ; u
.data:1001D991                  db  21h ; !
.data:1001D992                  db  3Dh ; =
.data:1001D993                  db  3Ch ; <
.data:1001D994                  db  26h ; &
.data:1001D995                  db  75h ; u
.data:1001D996                  db  37h ; 7
.data:1001D997                  db  34h ; 4
.data:1001D998                  db  36h ; 6
.data:1001D999                  db  3Eh ; >
.data:1001D99A                  db  31h ; 1
.data:1001D99B                  db  3Ah ; :
.data:1001D99C                  db  3Ah ; :
.data:1001D99D                  db  27h ; '
.data:1001D99E                  db  79h ; y
.data:1001D99F                  db  75h ; u
.data:1001D9A0                  db  26h ; &
```

应该是发现了一个新的区段，或许是代码，或许是其他函数的数据区段。从前面的一堆0来看，这里是一个独立新大陆。在这里运行IDApy




```
.data:1001D988 a1UUU7461Yu2u10 db 'xdoor is this backdoor, string decoded for '
.data:1001D9B3 aPracticalMalwa db 'Practical Malware Analysis Lab :)1234',0
```

注意提示，一定要在1001D988处

问题21：py代码如何工作的

```
1  sea = ScreenEA()
2
3  for i in range(0x00,0x50):
4          b = Byte(sea+i)
5          decoded_byte = b ^ 0x55
6          PatchByte(sea+i,decoded_byte)
7
```

简单的xor加密解密