

南開大學

恶意代码分析与防治技术实验报告

Lab6



学院：网络空间安全学院

专业：信息安全、法学

学号：2113203

姓名：付政烨

班级：信安法班

摘要

在恶意代码分析中，我们深入研究了一个代码样本的主要结构和功能。该代码首先在 0x401000 处的 if 语句中检查可用的 Internet 连接，然后在 0x40105F 处的 printf 子例程中输出信息。主要功能位于 0x401040，它下载指定网页并解析 HTML 注释，允许恶意代码获取指令和控制信息。一个新的函数位于 0x401130，通过 switch 语句执行各种操作，如错误信息输出、文件操作、注册表项设置和休眠。特定特征如 User-Agent 字段和 URL 可用于检测，而注册表键和文件路径作为本地特征进一步强化了检测。这种深入的恶意代码分析对于威胁检测和网络安全至关重要，帮助了解和应对潜在的恶意行为。

关键字：恶意代码分析； HTML 注释解析； 用户代理字段

目录

一、 实验目的	1
二、 实验原理	1
三、 实验过程	1
(一) Lab 6-1	1
(二) Lab6-2	3
(三) Lab 6-3	7
(四) Lab6-4	10
四、 Yara 规则编写	13
(一) Lab06-01.exe	13
(二) Lab06-02.exe	14
(三) Lab06-03.exe	15
(四) Lab06-04.exe	16
(五) 运行结果	16
五、 IDA Python 编写	17
六、 实验结论及心得体会	18

一、 实验目的

本实验旨在通过对一个二进制文件的分析,深入了解恶意代码的工作原理和行为。具体来说,实验的目的包括分析恶意代码的主要代码结构,包括位于 0x401000 处的 if 语句和位于 0x40105F 处的 printf 子例程,以理解其基本逻辑和功能。此外,实验旨在揭示恶意代码的主要行为,包括检查可用的 Internet 连接、下载网页并解析 HTML 注释,以及根据 HTML 注释中的内容执行本地操作,如删除文件、创建目录、设置注册表项等。在实验过程中,还运用了 YARA 规则等工具和方法,用于检测二进制文件的恶意特征。总而言之,这个实验旨在培养恶意代码分析的能力,帮助分析人员深入理解和检测潜在威胁,以提高计算机安全和恶意软件研究的技能。通过实际分析和理解恶意代码,可以更好地保护计算机系统和网络免受潜在威胁的侵害。

二、 实验原理

该实验的主要原理涉及对一个二进制程序进行逆向分析,以揭示其功能和操作。首先,程序检查是否存在可用的 Internet 连接,然后尝试下载一个包含 HTML 注释的网页。程序会解析 HTML 注释,其中的第一个字符决定了程序在本地系统上执行的操作,这些操作可能包括文件删除、目录创建、注册表设置、文件复制或休眠等。关键原理包括网络连接检测、HTTP 请求设置、HTML 注释解析、Switch 语句操作选择以及本地操作执行。通过观察程序的行为和操作,可以定义特征来识别潜在的恶意代码。这些特征包括特定字符串、User-Agent 字段、HTML 注释标记等。总之,这个实验旨在在深入了解和检测潜在威胁,通过分析程序的行为和特征来提高对恶意代码的识别能力。

三、 实验过程

(一) Lab 6-1

1. 由 main 函数调用的唯一子过程中发现的主要代码结构是什么?

通过 IDA 进行逆向分析,如图所示:

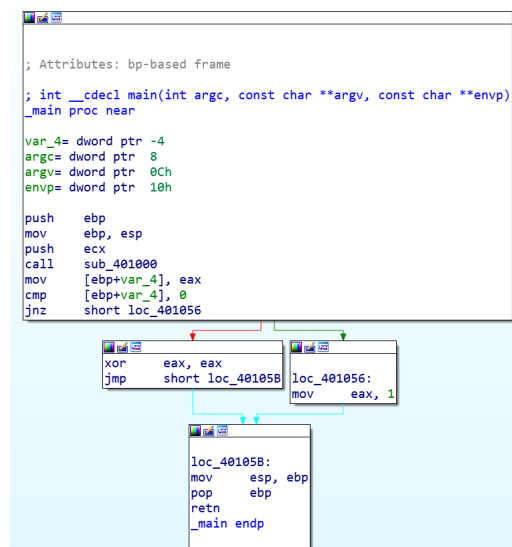


图 1

在程序的主函数 (main) 结构中, 可观察到以下主要过程:

1. 程序首先声明了一个名为 var_4 的局部变量。
2. 随后, 程序调用了 InternetGetConnectedState 函数。
3. 接着, 包含一个条件语句的分支结构, 以处理 InternetGetConnectedState 函数的返回结果, 即"success" 和"error"。
4. 若 InternetGetConnectedState 函数的返回值为 0, 程序将跳转至地址 loc_10102B。
5. 如果 InternetGetConnectedState 函数的返回值为 1, 程序将不执行跳转操作, 而是继续执行后续的指令。

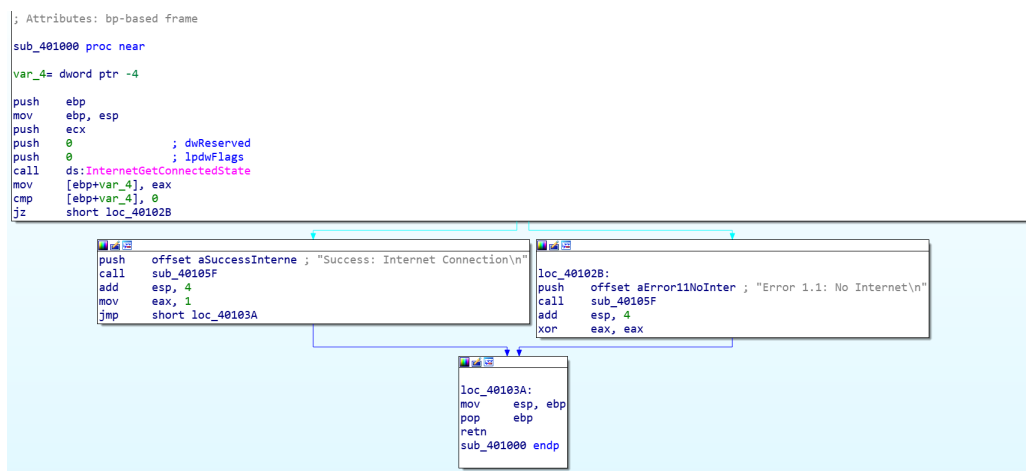


图 2

2. 位于 0x40105F 的子过程是什么?

```

sub_40105F proc near
    arg_0= dword ptr 4
    arg_4= dword ptr 8

    push    ebx
    push    esi
    mov     esi, offset stru_407098
    push    edi
    push    esi
    call    __stbuf
    mov     edi, eax
    lea     eax, [esp+10h+arg_4]
    push    eax ; int
    push    [esp+14h+arg_0] ; int
    push    esi ; FILE *
    call    sub_401282
    push    esi
    push    edi
    mov     ebx, eax
    call    __ftbuf
    add     esp, 18h
    mov     eax, ebx
    pop     edi
    pop     esi
    pop     ebx
    retn
sub_40105F endp

```

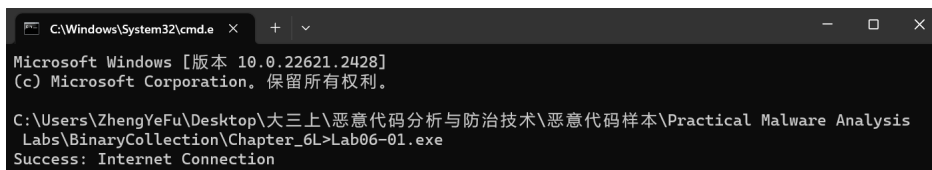
图 3

根据上图, 当出现错误时, 程序会执行相应的指令。在这一过程中, 程序按照顺序调用了三个函数, 并引入了一个名为"stru_407098"的数据偏移量, 其中包含了与"FILE" 字符相关的信

息。这个数据被存储在寄存器 esi 中，并在这三个函数的调用中被使用。值得注意的是，在进行 if 条件判断之前，程序在栈中先推送了一段字符串。因此，可以合理推测“sub_40105F” 函数是一个 printf 函数，用于格式化输出。

3. 这个程序的目的是什么？

该程序的主要目标在于验证当前系统是否具备可用的互联网连接。当程序确认互联网连接存在时，其输出将呈现为“Success: Internet Connection”，而在检测到缺乏互联网连接时，则会输出“Error 1.1: No Internet”。为验证该猜想，可通过执行命令行测试来进行实际检验。经过测试，发现实际情况符合我们的猜想。



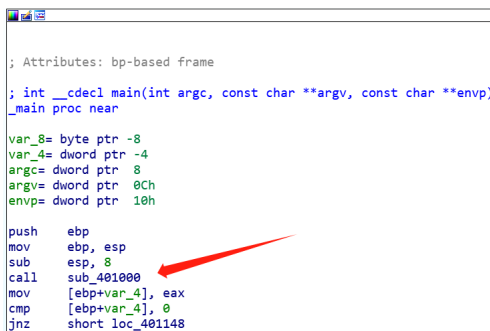
```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.22621.2428]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\ZhengYeFu\Desktop\大三上\恶意代码分析与防治技术\恶意代码样本\Practical Malware Analysis Labs\BinaryCollection\Chapter_6\Lab06-01.exe
Success: Internet Connection
```

图 4

(二) Lab6-2

1.main 函数调用的第一个子过程执行了什么操作？

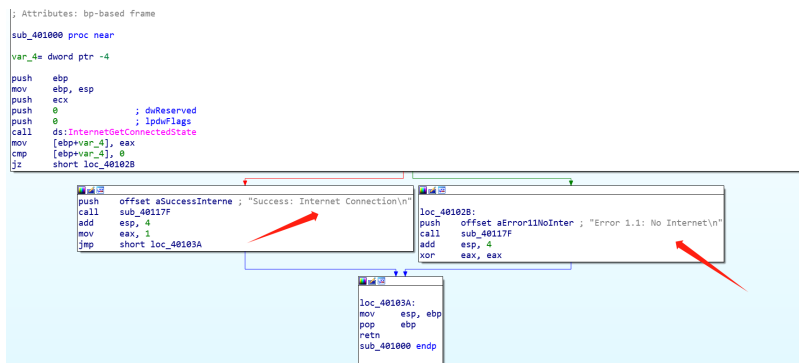


```
; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near
var_8= byte ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
sub     esp, 8
call    sub_401000
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jnz     short loc_401148
```

图 5

根据上图的展示，主函数在其执行中首先调用了名为 sub_401000 的子程序。通过双击进入 sub_401000 函数以查看其代码内容，我们得到以下结果：



```
; Attributes: bp-based frame
sub_401000 proc near
var_4= dword ptr -4
push    ebp
mov     ebp, esp
push    ecx
push    0
push    0
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_401028

loc_40117F:
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A

loc_401028:
push    offset aError11NoInter ; "Error 1.1: No Internet\n"
call    sub_40117F
add     esp, 4
xor     eax, eax

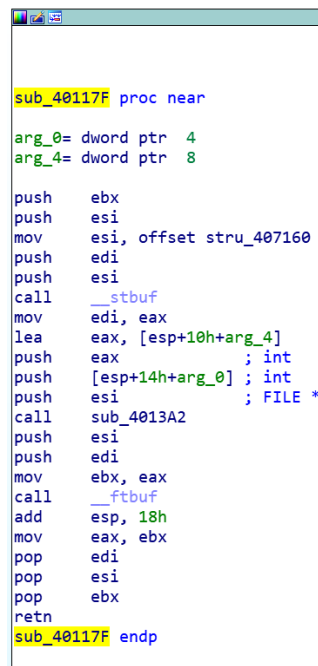
loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

图 6

在该函数内，发现其功能与 Lab06-01.exe 程序中的功能相同，即用于检查是否存在可用的网络连接。若当前设备处于联网状态，该函数将输出字符串“Success: Internet Connection”，而若设备未联网，则将输出字符串“Error 1.1: No Internet”。

2. 位于 0x40117F 的子过程是什么？

在对 sub_40117F 函数的内部进行审查时，我们观察到它与 Lab06-01.exe 中的 sub_40105F 函数存在相似性，即均为 printf 函数。



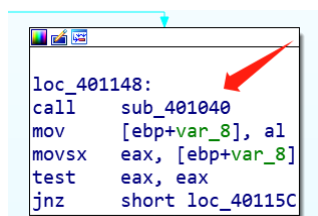
```
sub_40117F proc near
arg_0= dword ptr 4
arg_4= dword ptr 8

push    ebx
push    esi
mov     esi, offset stru_407160
push    edi
push    esi
call    __stbuf
mov     edi, eax
lea     eax, [esp+10h+arg_4]
push    eax ; int
push    [esp+14h+arg_0] ; int
push    esi ; FILE *
call    sub_4013A2
push    esi
push    edi
mov     ebx, eax
call    __ftbuf
add     esp, 18h
mov     eax, ebx
pop     edi
pop     esi
pop     ebx
retn
sub_40117F endp
```

图 7

3. 被 main 函数调用的第二个子过程做了什么？

如图所示，主函数中的第二个子例程是 sub_401040 函数。



```
loc_401148:
call    sub_401040
mov     [ebp+var_8], al
movsx   eax, [ebp+var_8]
test    eax, eax
jnz     short loc_40115C
```

图 8

通过双击进入函数内部，我们可以观察到多个与网络相关的函数的调用，以及 cmp（比较）和 jnz（条件跳转）指令，还有一些字符比较操作。

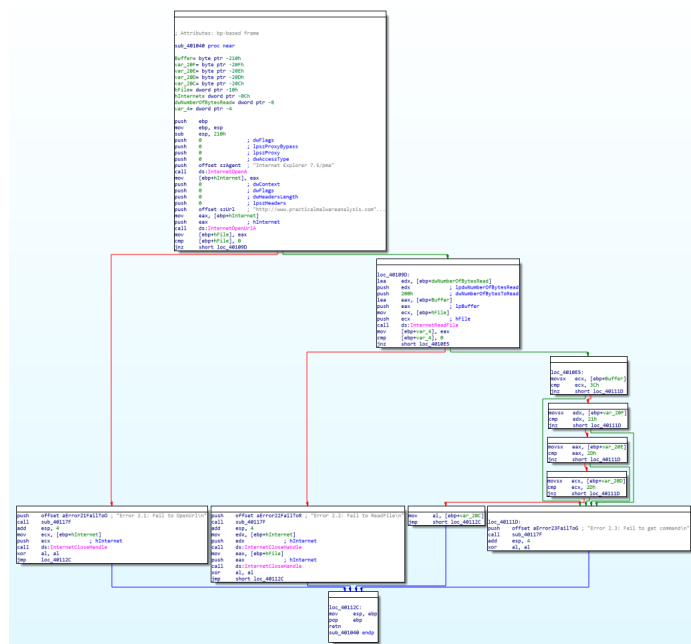


图 9

将注意力集中在字符比较操作，并使用“R”键来详细分析其内容。

```

.text:004010E5      movsx ecx, [ebp+Buffer]
.text:004010EC      cmp ecx, '<!'
.text:004010EF      jnz short loc_40111D
.text:004010F1      movsx edx, [ebp+var_20F]
.text:004010F8      cmp edx, '!'
.text:004010FB      jnz short loc_40111D
.text:004010FD      movsx eax, [ebp+var_20E]
.text:00401104      cmp eax, '-'
.text:00401107      jnz short loc_40111D
.text:00401109      movsx ecx, [ebp+var_20D]
.text:00401110      cmp ecx, '-'
.text:00401113      jnz short loc_40111D
.text:00401115      mov al, [ebp+var_20C]
.text:00401118      jmp short loc_40111C

```

图 10

我们发现字符比较操作是针对字符串“<!”进行的,这个字符串通常用于 HTML 文档的注释开始部分.综合以上分析,我们可以得出以下结论:sub_401040 函数首先调用 InternetOpenA 函数来打开网络连接,然后调用 InternetOpenUrlA 函数从“http://www.practicalmalwareanalysis.com”下载 HTML 网页.接下来的 cmp 指令用于进行判断,如果下载不成功,将打印字符串“Error 2.1: Fail to OpenUrl”并调用 InternetCloseHandle 函数来关闭连接.如果下载成功,将跳转到地址 loc_40109D, 然后调用 InternetReadFile 函数来读取已下载的 HTML 文件.紧接着, 又有一个 cmp 和 jnz 的比较跳转,如果读取不成功,将打印“Error 2.2: Fail to ReadFile”并调用 InternetCloseHandle 函数来关闭连接.如果读取成功,将跳转到地址 loc_4010E5, 执行代码以解析网页。

在网页解析过程中,我们可以看到一个解析规则,首先比较前四个字符是否为“<!”。如果所有比较都成功,就将第五个字符存储在寄存器 al 中,并作为返回值.如果这四个字符中任何一个比较失败,则将打印字符串“Error 2.3: Fail to get command”。因此,这个过程的目标是定位到网页注释正文的起始位置,并返回相应的首地址。

4. 在这个子过程中使用了什么类型的代码结构？

该子例程在执行过程中使用 InternetReadFile 函数从网络获取数据，并将获取的数据存储到一个字符数组中。随后，它逐字节遍历该数组，逐一比对以解析 HTML 注释。

5. 在这个程序中有任何基于网络的指示吗？

如下图所示，该程序主要存在两项网络特征。所述程序采用了 Internet Explorer 7.5 版本的 User-Agent 字段作为 HTTP 请求标识，并通过 HTTP 协议从网址 <http://www.practicalmalwareanalysis.com/cc.htm> 下载了一个网页页面。

```
.data:00407030 aError11NoInter db 'Error 1.1: No Internet',0Ah,0
.data:00407030 ; DATA XREF: sub_401000:loc_40102Bf0
.data:00407048 aSuccessInterne db 'Success: Internet Connection',0Ah,0
.data:00407048 ; DATA XREF: sub_401000+17f0
.data:00407066 align 4
.data:00407068 aError23FailToG db 'Error 2.3: Fail to get command',0Ah,0
.data:00407068 ; DATA XREF: sub_401040:loc_40111Df0
.data:00407088 aError22FailToR db 'Error 2.2: Fail to ReadFile',0Ah,0
.data:00407088 ; DATA XREF: sub_401040+80f0
.data:004070A5 align 4
.data:004070A8 aError21FailToO db 'Error 2.1: Fail to OpenUrl',0Ah,0
.data:004070A8 ; DATA XREF: sub_401040+3Ff0
.data:004070C4 ; CHAR szUrl[]
.data:004070C4 db 'http://www.practicalmalwareanalysis.com/cc.htm',0
.data:004070C4 ; DATA XREF: sub_401040+27f0
.data:004070F3 align 4
.data:004070F4 ; CHAR szAgent[]
.data:004070F4 db 'Internet Explorer 7.5/pma',0
.data:004070F4 ; DATA XREF: sub_401040+11f0
.data:0040710E align 10h
.data:00407110 aSuccessParsedC db 'Success: Parsed command is %c',0Ah,0
```

图 11

6. 这个恶意代码的目的是什么？

在 main 函数的结构中，经过 sub_401040 函数的调用后，经过解析处理的字符的首地址会被存储在寄存器 al 中，并随后将其传递至寄存器 ecx，然后作为参数被推入栈中。接着，程序将输出字符串“Success: Parsed command is %c”，其中 %c 代表的是 sub_401040 函数解析出的字符串。紧接着，程序将休眠 0EA60h（相当于 60000 毫秒，即 60 秒）。

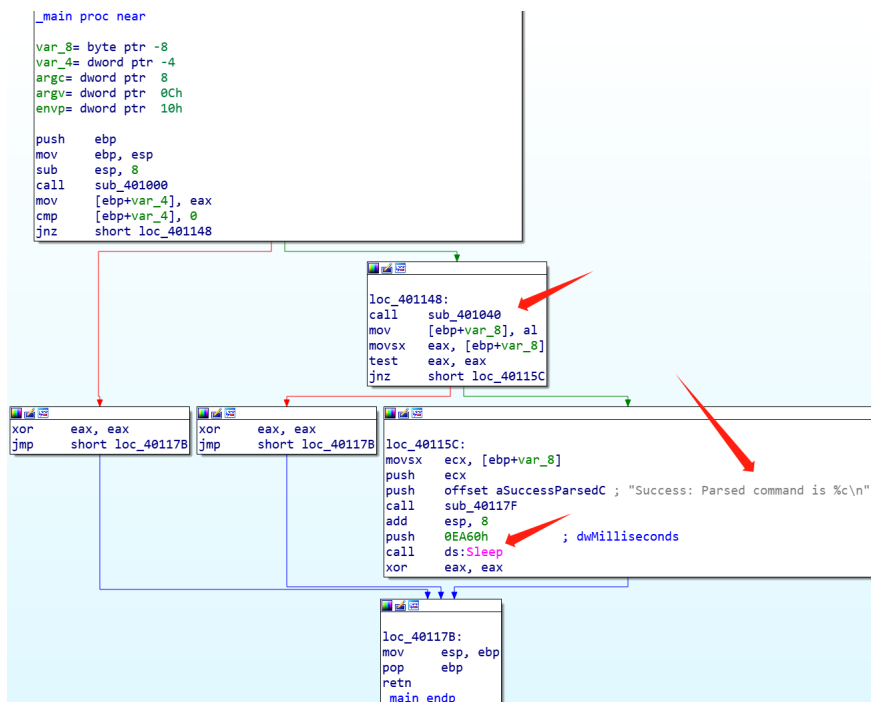


图 12

进一步分析可得, 该程序首先会检查是否存在可用的网络连接, 如果没有可用网络连接, 则程序将停止执行。若网络已连接, 则程序将下载并解析 <http://www.practicalmalwareanalysis.com> 网址的 HTML 文件, 接着输出字符串 “Success: Parsed command is”, 随后附加注释正文字符。最后, 程序将休眠 60 秒, 然后终止运行。

(三) Lab 6-3

1. 比较在 main 函数与实验 6-2 的 main 函数的调用。从 main 中调用的新的函数是什么?

在内存地址范围 0x401000 至 0x401040 之间的函数与 Lab 6-2 中的函数相同。而在 0x401271 处, 我们观察到了一个 printf 函数的存在。至于内存地址 0x401130 处的函数, 则是本次实验中首次出现的。

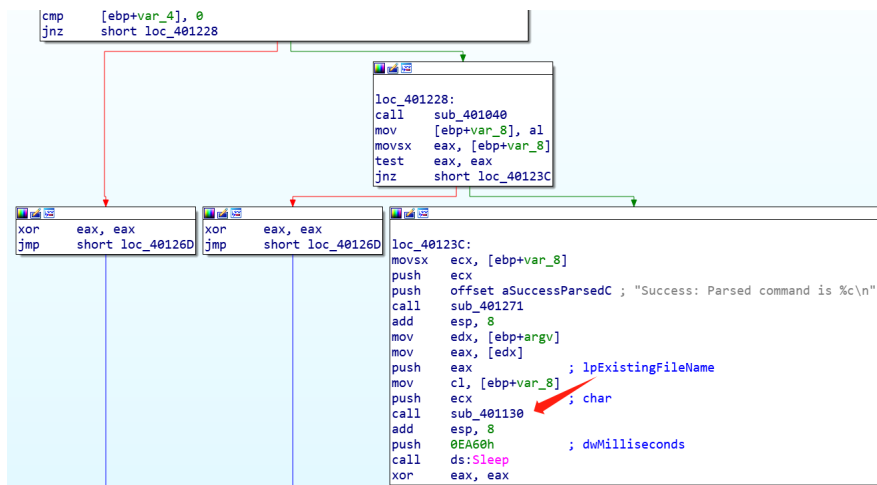


图 13

2. 这个新的函数使用的参数是什么?

在函数调用过程中, 先后将寄存器 eax 和 ecx 入栈。根据函数栈帧的结构, 此函数接受两个参数。从源代码的逻辑可以观察到, 将存储在 argv 地址中的值加载到了 eax 寄存器中。而 argv 的地址即指向 main 函数中的 argv[0], 这在本例中是一个指向程序的路径和名称的指针。因此, eax 中包含的是程序的名称, 即字符串 “Lab06-03.exe”。



图 14

至于 ecx 的分析如下图所示，它存储了 var_8 的内容，而 var_8 的内容实际上是 al，也就是 sub_401040 函数的返回值。前面已经提到，sub_401040 函数的返回值是所下载 HTML 文件的注释正文的首地址。

```
.text:00401254      mov     cl, [ebp+var_8]
.text:00401257      push   ecx                ; char
.text:00401258      call   sub_401130
.text:0040125D      add     esp, 8
```

图 15

因此，这两个参数分别是程序的名称和注释正文的字符。

3 这个函数包含的主要代码结构是什么？

根据该函数的结构框架，存在五个跳转点。

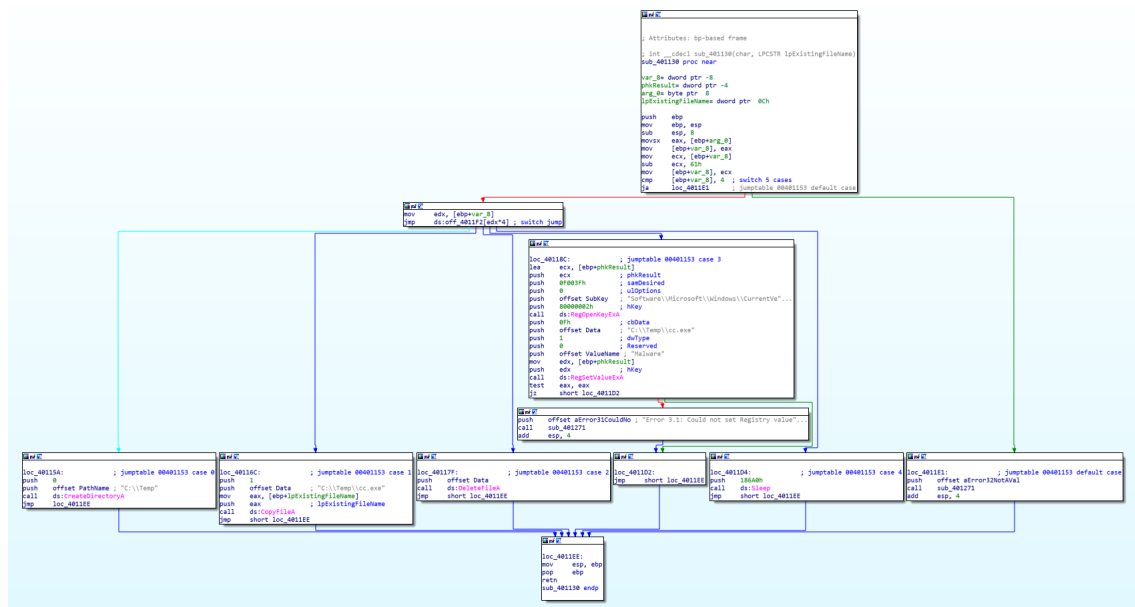


图 16

在此处，存在一个跳转表，其跳转的条件基于参数 var_8 的地址处的内容。具体而言，如果 var_8 的内容为：

```
.text:004011F1 ;-----
.text:004011F2 dd offset loc_4011F2
.text:004011F2
.text:004011F2
.text:004011F2
.text:004011F2
.text:00401206
                align 10h

                ; DATA XREF: sub_401130+231r
                ; jump table for switch statement
dd offset loc_40115A
dd offset loc_40116C
dd offset loc_40117F
dd offset loc_40118C
dd offset loc_4011D4
```

图 17

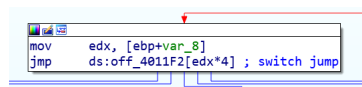


图 18

- "a"，则会跳转到标签 loc_40115A；

- "b", 则会跳转到标签 loc_40116C;
- "c", 则会跳转到标签 loc_40117F;
- "d", 则会跳转到标签 loc_40118C;
- "e", 则会跳转到标签 loc_4011D4;
- 其它数据, 将跳转到标签 loc_4011E1。此部分已经被鉴别为一个 switch 语句结构。

4 这个函数能够做什么？

1. 当参数被设置为'a' 时, 程序在跳转到地址 loc_40115A 时, 它执行以下操作: 创建了目录'C:\ Temp'。
2. 当参数被设置为'b' 时, 程序在跳转到地址 loc4_40116C 时, 它执行以下操作: 调用了 CopyFileA 函数, 并在调用前推入了两个参数, 其中一个是字符串 "C:\ Temp\ cc.exe", 另一个是 lpExistingFileName 的内容。lpExistingFileName 是 sub4_401130 函数的第二个参数, 在函数调用前, 程序先将 argv 压入栈, 这个参数包含了程序本身的名称。因此, 当程序跳转到 loc4_40116C 时, 它实际上是在将程序本身的名称复制到'C:\ Temp' 目录下, 并将其重命名为'cc.exe'。
3. 当参数被设置为'c' 时, 程序在跳转到地址 loc4_40117F 时, 它执行以下操作: 删除'C:\ Temp\ cc.exe' 文件。
4. 当参数被设置为'd' 时, 程序在跳转到地址 loc_40118C 时, 执行以下操作: 调用 RegOpenKeyExA 函数来打开注册表路径'Software\ Microsoft\ Windows\ CurrentVersion\ Run', 然后使用 RegSetValueExA 函数将'C:\ Temp\ cc.exe' 的路径写入注册表, 并将其设置为开机启动项。
5. 当参数被设置为'e' 时, 程序在跳转到地址 loc_4011D4 时, 执行以下操作: 让程序休眠 186A0h 毫秒, 相当于 100 秒的延时。

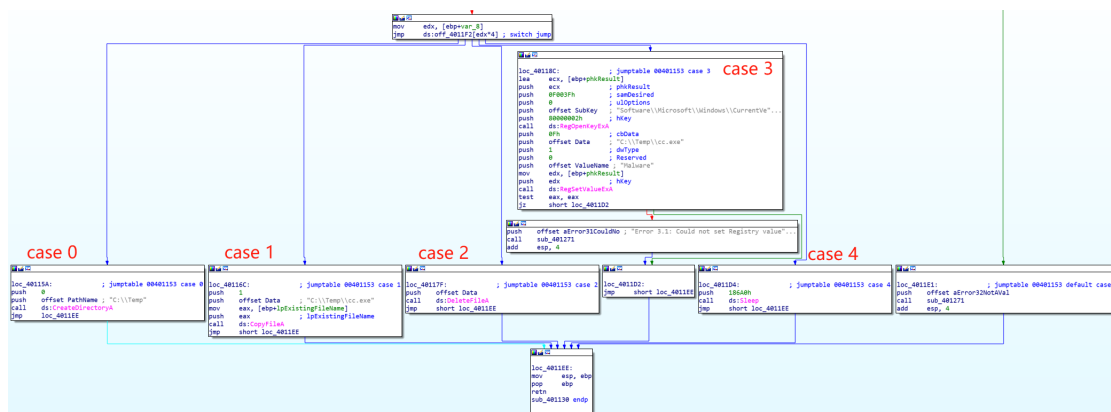


图 19

5. 在这个恶意代码中有什么本地特征吗？

打开字符串窗口，可观察到本地文件路径，并基于之前的分析结果，得知该恶意代码以本地特征的方式，首先生成了名为 C:\Temp 的目录，并在该目录下创建了 cc.exe 程序。随后，该恶意代码对系统注册表进行了修改，将 C:\Temp\cc.exe 设置为系统开机自启动的项。

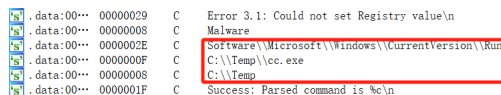


图 20

6. 这个恶意代码的目的是什么？

根据上述分析，我们可以得出如下结论：该恶意代码首先执行一个联网检测操作，如果检测到未连接到互联网，它将立即停止执行；而如果已经建立了互联网连接，该代码将下载一个 HTML 文件，并随后对其进行解析。根据所下载 HTML 文件中注释的首字母，即“a, b, c, d, e”，该恶意代码将自我复制到 C:\Temp 路径，并设置自己为系统开机自启动项。

(四) Lab6-4

1. 在实验 6-3 和 6-4 的 main 函数中的调用之间的区别是什么？

对 main 函数的代码结构进行观察，得到如下结果：

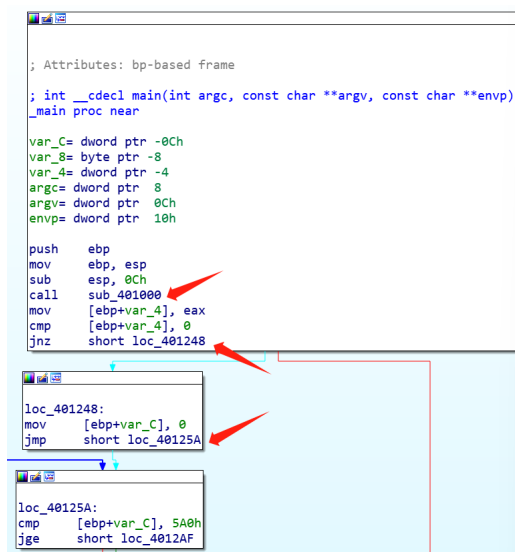


图 21

首先，main 函数开始时调用 sub_401000 函数。类似之前的实验，此函数的目的是检测联网状态，如果已联网，则程序跳转至标记 loc_401248。

在标记 loc_401248 处，可以观察到这部分代码与之前的实验相比更加复杂。它执行了将位于地址 var_C 偏移处的内容设为零的操作，然后无条件跳转至标记 loc_40125A。随后，继续执行标记 loc_40125A 处的代码块，这部分代码与 Lab06-03.exe 相比多出了一些内容。在这里，与数值 5A0 进行比较，并使用 jge 指令来检查大小关系。由于之前的变量 Var_C 已被赋值为零，显然无法满足跳转条件，因此代码继续向下执行。

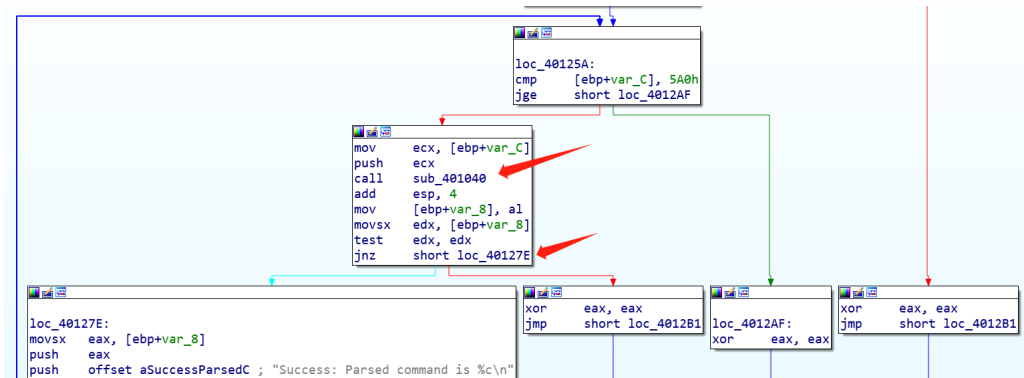


图 22

接下来的代码调用了 sub_401040 函数, 与之前的实验相似, 其目的是下载 HTML 文件并返回注释正文的首地址。如果成功返回首地址, 程序将跳转至标记 loc_40127E。在标记 loc_40127E 处, 与之前的实验内容相符, 首先打印注释正文的第一个字符, 然后调用 sub_401150 函数, 最后程序休眠 60 秒, 之后无条件跳转至标记 loc_401251 处。

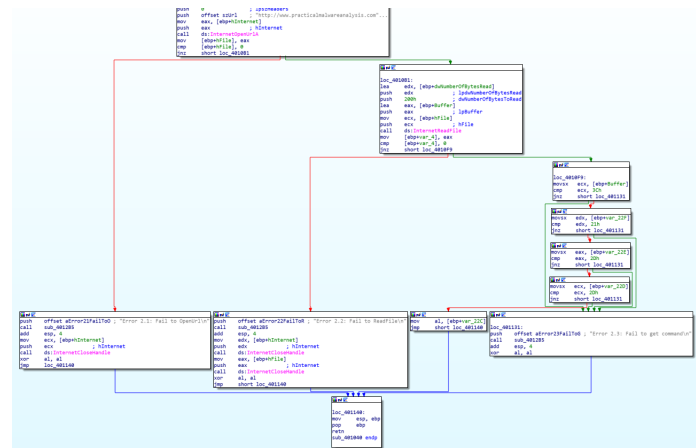


图 23

sub_401150 函数的结构如上图所示, 实际上具有与之前上一小节分析的 switch 函数相同的功能。

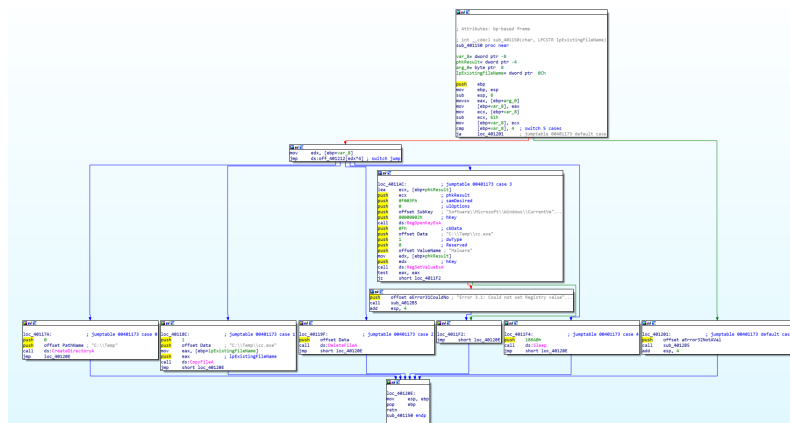


图 24

在标记 loc_401251 处，可以观察到对变量 var_C 执行加 1 的操作，类似于 C 语言中的自增操作。然后程序再次回到标记 loc_40125A，继续与 5A0h 进行比较。明显可见，这是一个循环结构，一直循环直到变量大于或等于 5A0h 时才跳出循环并结束程序。

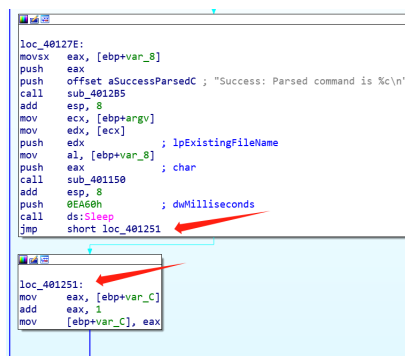


图 25

2. 什么新的代码结构已经被添加到 main 中?

在主函数中嵌入了一个 for 循环语句。

3. 这个实验的解析 HTML 的函数和前面实验中的那些有什么区别?

在调用解析函数之前，该程序首先将用于表示循环次数的变量推送到栈上，并将其作为 sub_401040 函数的参数。以下为关键代码区示例。

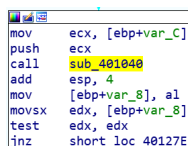


图 26

在进入 sub_401040 函数后，可以观察到与之前的字符串 "Internet Explorer 7.50/pma%d" 相比，发生了变化，它现在包含一个额外的 %d 类型参数，这个参数对应于变量 arg_0，即在函数调用之前传递的参数，即循环次数。接下来，程序调用了 sprintf 函数，用于格式化传入的字符串，然后将结果传递给 InternetOpenA 函数。



图 27

4. 这个程序会运行多久?(假设它已经连接到互联网)

根据程序中的 sleep 函数调用,以观察程序的休眠时长。首先,在函数 sub_401150 执行完毕后,存在一个 0xEA60h 毫秒的休眠,即 60 秒的等待时间。此休眠时长代表每一次迭代的等待时间,如前文所分析,总共需要进行 0x5A0 次迭代,相当于 1440 次。因此,每个迭代需要 1 分钟,总程序运行时间为 1440 分钟,即 24 小时。

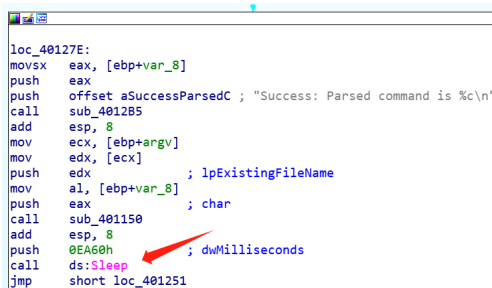


图 28

在 sub_401150 函数内,如果 switch 结构的参数为'e',同样会导致程序休眠 0x186A0h 毫秒,即 100 秒,因此,程序每次运行至少需要 24 小时。

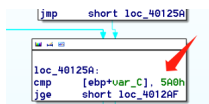


图 29

5. 在这个恶意代码中有什么新的基于网络的迹象吗?

是的,采用了一种新的用户代理 (User-Agent) 标识。其格式为 Internet Explorer 7.50/pma%d,其中%d 表示已经执行的程序运行时间 (以分钟为单位)。

6. 这个恶意代码的目的是什么?

首先,该程序会进行首要检查,以核实是否存在可用的互联网连接。在未能找到可用连接的情况下,程序将立即中止其执行。然而,若存在可用互联网连接,程序随即采用 b 个不同的独特 User-Agent 标识,用于请求下载一个网页。这些 User-Agent 标识中嵌入了一个计时器,用于追踪程序已经运行的时间,以分钟为单位。

所下载的网页包含以 HTML 注释代码 <!--开头的部分。在此注释代码中,紧随其后的第一个字符用于触发一个 switch 语句,该语句用于决定在本地系统上应采取何种行为。这些行为包括一系列硬编码操作,例如文件删除、目录创建、注册表 run 键的设置、文件复制以及 100 秒的休眠等操作。程序会持续运行 24 小时,然后终止其运行。

四、 Yara 规则编写

(一) Lab06-01.exe

```
1 rule Lab06_01 {
2   meta:
```



```

3      description = "Rule for detecting Lab06-01.exe"
4      strings:
5          $success = "Success: Internet Connection" fullword ascii
6          $error = "Error 1.1: No Internet" fullword ascii
7          $function = "InternetGetConnectedState" fullword ascii
8      condition:
9          uint16(0) == 0x5a4d and
10         uint32(uint32(0x3c)) == 0x00004550 and
11         filesize < 100KB and
12         all of them
13 }

```

这个 YARA 规则的功能是检测 Lab06-01.exe 文件是否包含与特定的恶意或恶意行为相关的字符串和 PE 文件头特征。以下是规则的功能描述：

- description 元数据字段提供了规则的描述，即它是用于检测 Lab06-01.exe 文件的规则。
- 规则定义了三个字符串变量 \$s1, \$s2, 和 \$s3，分别包含了特定的 ASCII 字符串，这些字符串与成功的互联网连接、互联网断开的错误和“InternetGetConnectedState”函数相关。
- 规则要求文件的前两个字节是“0x5a4d”，这指示它是一个 PE 文件。
- 规则还要求在文件指针偏移 0x3c 处的值为“0x00004550”，以确认 PE 文件的合法性。
- 文件大小必须小于 100KB，以限制规则的应用范围。
- 规则的 condition 部分要求文件中必须包含规则中定义的所有三个字符串，这样才能被视为 Lab06-01.exe 的潜在恶意文件。

(二) Lab06-02.exe

```

1 rule Lab06_02 {
2     meta:
3         description = "Detection rule for Lab06-02.exe"
4     strings:
5         $errorNoInternet = "Error 1.1: No Internet" fullword ascii
6         $errorOpenUrlFail = "Error 2.1: Fail to OpenUrl" fullword ascii
7         $successParsedCommand = "Success: Parsed command is %c" fullword ascii
8         $errorReadFileFail = "Error 2.2: Fail to ReadFile" fullword ascii
9         $successInternetConnection = "Success: Internet Connection" fullword ascii
10        $errorGetCommandFail = "Error 2.3: Fail to get command" fullword ascii
11        $maliciousURL = "http://www.practicalmalwareanalysis.com/cc.htm" fullword
12                           ascii
13        $userAgent = "Internet Explorer 7.5/pma" fullword ascii
14    condition:
15        uint16(0) == 0x5a4d and
16        uint32(uint32(0x3c)) == 0x00004550 and filesize < 100KB and
17        6 of them
18 }

```

上述 YARA 规则用于检测 Lab06-02.exe 文件的恶意特征。规则功能如下：

- description 元数据字段提供了规则的描述，即它是用于检测 Lab06-02.exe 文件的规则。
- 规则通过检查文件中的特定字符串来识别恶意行为，包括错误消息、成功消息以及恶意 URL 和用户代理字符串。

- 规则要求文件的前两个字节是”0x5a4d”，这指示它是一个 PE 文件。
- 规则还要求在文件指针偏移 0x3c 处的值为”0x00004550”，以确认 PE 文件的合法性。
- 文件大小必须小于 100KB，以限制规则的应用范围。
- 最后的 condition 部分要求文件中至少包含规则中定义的六个特定字符串中的任意六个，以被视为 Lab06-02.exe 的潜在恶意文件。

(三) Lab06-03.exe

```
1 rule Lab06_03 {
2   meta:
3     description = "Detects Lab06-03.exe"
4   strings:
5     $maliciousFilePath = "C:\\Temp\\cc.exe" fullword ascii
6     $error2_3 = "Error 2.3: Fail to get command" fullword ascii
7     $error3_2 = "Error 3.2: Not a valid command provided" fullword ascii
8     $ccUrl = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
9     $successParsedCommand = "Success: Parsed command is %c" fullword ascii
10    $error2_2 = "Error 2.2: Fail to ReadFile" fullword ascii
11    $successInternetConnection = "Success: Internet Connection" fullword ascii
12    $error3_1 = "Error 3.1: Could not set Registry value" fullword ascii
13    $error1_1 = "Error 1.1: No Internet" fullword ascii
14    $error2_1 = "Error 2.1: Fail to OpenUrl" fullword ascii
15    $malwareIndicator = "Malware" fullword ascii /* Goodware String - occurred 12
        times */
16    $pmaString = "Internet Explorer 7.5/pma" fullword ascii
17   condition:
18     uint16(0) == 0x5a4d and
19     uint32(uint32(0x3c)) == 0x00004550 and filesize < 100KB and
20     1 of ($maliciousFilePath, $error2_3, $error3_2, $ccUrl, $successParsedCommand
        , $error2_2, $successInternetConnection, $error3_1, $error1_1, $error2_1)
        and
21     $malwareIndicator
22 }
```

上述 YARA 规则用于检测 Lab06-03.exe 文件的恶意特征。规则功能如下：

- description 元数据字段提供了规则的描述，即它是用于检测 Lab06-03.exe 文件的规则。
- 规则通过检查文件中的特定字符串来识别恶意行为，包括错误消息、成功消息、恶意 URL、用户代理字符串以及与恶意软件相关的标识字符串。
- 规则要求文件的前两个字节是”0x5a4d”，这指示它是一个 PE 文件。
- 规则还要求在文件指针偏移 0x3c 处的值为”0x00004550”，以确认 PE 文件的合法性。
- 文件大小必须小于 100KB，以限制规则的应用范围。
- 规则要求文件中至少包含规则中定义的十个特定字符串中的一个，以被视为 Lab06-03.exe 的潜在恶意文件。
- 最后一个条件 \$malwareIndicator 要求必须存在名为”Malware”的字符串，这是一个恶意软件标识符。

(四) Lab06-04.exe

```

1 rule Lab06_04 {
2     meta:
3         description = "Detect Lab06-04 Malware"
4     strings:
5         $ccExePath = "C:\\Temp\\cc.exe" fullword ascii
6         $error2_3 = "Error 2.3: Fail to get command" fullword ascii
7         $error3_2 = "Error 3.2: Not a valid command provided" fullword ascii
8         $ccHtmlUrl = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
9         $successParsedCommand = "Success: Parsed command is %c" fullword ascii
10        $error2_2 = "Error 2.2: Fail to ReadFile" fullword ascii
11        $successInternetConnection = "Success: Internet Connection" fullword ascii
12        $error3_1 = "Error 3.1: Could not set Registry value" fullword ascii
13        $error1_1 = "Error 1.1: No Internet" fullword ascii
14        $error2_1 = "Error 2.1: Fail to OpenUrl" fullword ascii
15        $malwareString = "Malware" fullword ascii
16    condition:
17        uint16(0) == 0x5a4d and
18        uint32(uint32(0x3c)) == 0x00004550 and filesize < 100KB and
19        1 of ($ccExePath, $error2_3, $error3_2, $ccHtmlUrl, $successParsedCommand,
20            $error2_2, $successInternetConnection, $error3_1, $error1_1, $error2_1)
21        and
22        all of them
23 }

```

上述 YARA 规则的功能是检测文件是否符合以下条件：

- 文件的开头必须以字节序列“0x5a4d”（表示 PE 文件格式）开始。
- 文件中的第一个节的偏移地址必须指向“0x00004550”（表示 PE 头的标识）。
- 文件大小必须小于 100KB。
- 文件中必须包含以下字符串之一：cc.exe 文件路径、特定错误信息、特定成功信息、特定 URL，以及其他相关字符串。
- 文件中必须包含所有这些特定字符串，即必须同时存在。

(五) 运行结果

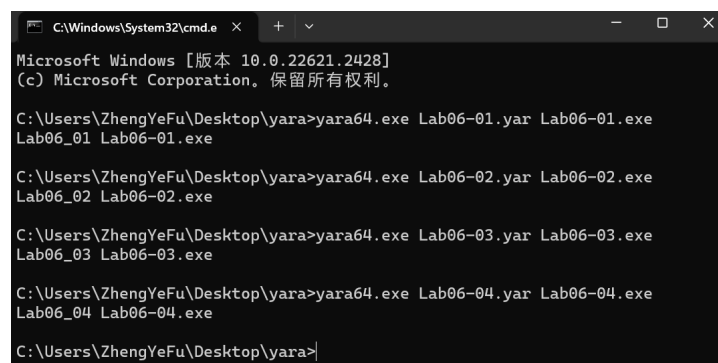


图 30

五、 IDA Python 编写

获取已知库函数信息、查看区段信息以及函数指令查看在恶意代码分析中具有重要意义。这些信息可帮助分析人员追踪外部依赖项、理解内存布局、定位潜在威胁，同时揭示恶意代码的操作和目的。综合利用这些工具，可提高分析的深度和准确性，有助于更有效地识别恶意功能和漏洞。为此，我编写了如下代码，可以实现获取已知库函数信息、查看区段信息和函数指令查看的功能：

```
1 rule Lab06_04 {
2     meta:
3         description = "Detect Lab06-04 Malware"
4     strings:
5         $ccExePath = "C:\\Temp\\cc.exe" fullword ascii
6         $error2_3 = "Error 2.3: Fail to get command" fullword ascii
7         $error3_2 = "Error 3.2: Not a valid command provided" fullword ascii
8         $ccHtmlUrl = "http://www.practicalmalwareanalysis.com/cc.htm" fullword ascii
9         $successParsedCommand = "Success: Parsed command is %c" fullword ascii
10        $error2_2 = "Error 2.2: Fail to ReadFile" fullword ascii
11        $successInternetConnection = "Success: Internet Connection" fullword ascii
12        $error3_1 = "Error 3.1: Could not set Registry value" fullword ascii
13        $error1_1 = "Error 1.1: No Internet" fullword ascii
14        $error2_1 = "Error 2.1: Fail to OpenUrl" fullword ascii
15        $malwareString = "Malware" fullword ascii
16    condition:
17        uint16(0) == 0x5a4d and
18        uint32(uint32(0x3c)) == 0x00004550 and filesize < 100KB and
19        1 of ($ccExePath, $error2_3, $error3_2, $ccHtmlUrl, $successParsedCommand,
20            $error2_2, $successInternetConnection, $error3_1, $error1_1, $error2_1)
21        and
22        all of them
23 }
```

这段代码是使用 IDA Pro 的 Python API 来获取有关当前二进制文件的信息并打印出来，以下是代码的功能介绍：

print_known_libraries 函数：

- 该函数用于获取当前二进制文件的已知库函数信息并打印出来。
- 使用 `idaapi.get_import_module_qty()` 获取已知库函数的数量。
- 遍历已知库函数列表，获取每个库函数的名称，并打印出来。

print_segments 函数：

- 该函数用于获取当前二进制文件中的区段信息并打印出来。
- 使用 `idaapi.get_segm_qty()` 获取区段（segment）的数量。
- 遍历每个区段，获取其起始地址和终止地址，并打印出来。

获取当前光标位置的地址：

- 使用 `idc.ScreenEA()` 获取当前光标位置的地址，并将其存储在 `current_address` 变量中。

打印已知库函数信息和查看的区段信息：

- 调用 `print_known_libraries()` 函数来打印已知库函数信息。
- 调用 `print_segments()` 函数来打印查看的区段信息。

获取地址所在的函数的起始地址：

- 使用 `idc.GetFunctionAttr(current_address, idc.FUNCATTR_START)` 获取包含当前地址的函数的起始地址，并将其存储在 `function_start` 变量中。

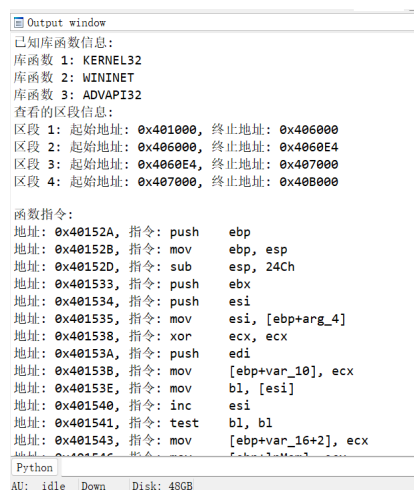
打印函数指令信息：

- 如果 `function_start` 不等于 `BADADDR`，表示当前地址在一个函数内，那么获取该函数的起始地址。
- 使用 `idc.FindFuncEnd(function_start)` 获取函数指令的结束地址。
- 遍历函数内的每个指令，获取其地址和反汇编文本，并打印出来。

处理地址不在任何函数内的情况：

- 如果 `function_start` 等于 `BADADDR`，则说明当前地址不在任何函数内，会输出一个相应的消息。

运行结果



```
Output window
已知库函数信息：
库函数 1: KERNEL32
库函数 2: WININET
库函数 3: ADVAPI32
查看的区段信息：
区段 1: 起始地址: 0x401000, 终止地址: 0x406000
区段 2: 起始地址: 0x406000, 终止地址: 0x4060E4
区段 3: 起始地址: 0x4060E4, 终止地址: 0x407000
区段 4: 起始地址: 0x407000, 终止地址: 0x408000

函数指令：
地址: 0x40152A, 指令: push    ebp
地址: 0x40152B, 指令: mov     ebp, esp
地址: 0x40152D, 指令: sub     esp, 24Ch
地址: 0x401533, 指令: push    ebx
地址: 0x401534, 指令: push    esi
地址: 0x401535, 指令: mov     esi, [ebp+arg_4]
地址: 0x401538, 指令: xor     ecx, ecx
地址: 0x40153A, 指令: push    edi
地址: 0x40153B, 指令: mov     [ebp+var_10], ecx
地址: 0x40153E, 指令: mov     bl, [esi]
地址: 0x401540, 指令: inc     esi
地址: 0x401541, 指令: test    bl, bl
地址: 0x401543, 指令: mov     [ebp+var_16+2], ecx
```

图 31

六、实验结论及心得体会

通过本次实验，我们深刻认识到在网络安全领域中，分析恶意代码的内部结构和功能对于检测和应对威胁具有至关重要的意义。通过静态分析恶意代码，我们不仅可以揭示其潜在威胁，还能了解其工作原理和行为方式。本次实验中，我们首先观察了恶意代码的主要结构，包括检查 Internet 连接的可用性，以确保网络连接正常运行，然后尝试下载特定网页。这个网页包含 HTML 注释，而注释的首字符将决定后续操作，如文件删除、目录创建、注册表项设置等。特别值得注意的是，新引入的函数使用了 `switch` 语句和跳转表，实现了多种操作，显示了恶意代码的多样性和操控能力。此外，我们还学习了如何使用 YARA 规则和特征识别来检测恶意代码的存在。

在进行恶意代码分析时，需要仔细观察代码结构和函数调用，借助 YARA 规则和特征识别来迅速识别威胁。同时，了解恶意代码的潜在威胁和构建有效的安全机制对于提高网络安全至关重要。通过不断学习和实践，我们可以提高对恶意代码的识别和应对能力，从而更好地保护系统和数据的安全。这次实验为我们提供了宝贵的学习经验，使我们更深刻地理解了恶意代码分析的流程和工具，以及在网络安全领域的应用重要性。

参考文献

- [1] SM-D.Practical Malware Analysis[J].Network Security, 2012, 2012(12):4-4.DOI:10.1016/S1353-4858(12)70109-5.