windows样本高级静态分析之识别汇编中C代码结构三

yong夜 / 2019-10-16 09:20:25 / 浏览数 3028 安全技术 二进制安全 顶(0) 踩(0)

目标

通过分析代码结构来理解一个恶意样本的总体功能。

本篇主要通过分析样本了解for、while流程语句

分析流程

1.基础静态分析

2.基础动态分析

3.高级静态分析

实践过程

实例1

Lab06-04.exe

基础静态分析

导入函数

```
InternetOpenUrlA
InternetCloseHandle
InternetReadFile
InternetGetConnectedState
InternetOpenA
RegSetValueExA
RegOpenKeyExA
CreateDirectoryA
CopyFileA
DeleteFileA
GetFileType
WriteFile
```

字符串

```
http://www.practicalmalwareanalysis.com/cc.htm
Software\Microsoft\Windows\CurrentVersion\Run
C:\Temp\cc.exe
C:\Temp
Error 1.1: No Internet
Success: Internet Connection
Error 2.3: Fail to get command
Error 2.2: Fail to ReadFile
Error 2.1: Fail to OpenUrl
Internet Explorer 7.50/pma%d
Error 3.2: Not a valid command provided
Error 3.1: Could not set Registry value
Malware
Success: Parsed command is %c
```

这里和上一篇文章大同小异，主要在Internet Explorer 7.50/pma后面多了一个占位符

根据api和字符串可以判断：

1.存在联网访问http://www.practicalmalwareanalysis.com/cc.htm 网址操作并且通过字符串中的错信息可以判断可能存在解析网页来获取命令来执行

2.写注册表来是实现自启动

3.产生衍生文件C:\Temp\cc.exe

基础动态分析

```
C:\Users\15pb-win7\Desktop\Chapter_6L>C:\Users\15pb-win7\Desktop\Chapter_6L\Lab06-04.exe
Success: Internet Connection
Error 2.1: Fail to OpenUrl

C:\Users\15pb-win7\Desktop\Chapter_6L>_
```


和之前分析一样，根据不同网络状态返回打印内容，接着通过高级静态分析来看程序后续操作

高级静态分析

跟进main方法分析，大部分和Lab06-03.exe相同，下面主要分析不同之处

for循环流程

for循环主要包括有：初始化、判断条件、条件成立后执行的语句块、语句块执行完毕后的递增或递减

所以在汇编指令中存在3个跳转：

 1.初始化完毕后跳转到判断条件

 2.判断条件不成立引起的跳出for循环

 3.条件成立后执行完语句块后跳到递增或递减的语句块处

```
.text:00401230
.text:00401230                 push    ebp
.text:00401231                 mov     ebp, esp
.text:00401233                 sub     esp, 0Ch
.text:00401236                 call    network_status
.text:0040123B                 mov     [ebp+var_4], eax
.text:0040123E                 cmp     [ebp+var_4], 0
.text:00401242                 jnz     short loc_401248
.text:00401244                 xor     eax, eax
.text:00401246                 jmp     short loc_4012B1
.text:00401248 ; ---------------------------------------------------------------------------
.text:00401248
.text:00401248 loc_401248:                             ; CODE XREF: _main+12↑j
.text:00401248                 mov     [ebp+var_C], 0
.text:0040124F                 jmp     short loc_40125A
.text:00401251 ; ---------------------------------------------------------------------------
.text:00401251
.text:00401251 loc_401251:                             ; CODE XREF: _main+7D↓j
.text:00401251                 mov     eax, [ebp+var_C]
.text:00401254                 add     eax, 1
.text:00401257                 mov     [ebp+var_C], eax
.text:0040125A
.text:0040125A loc_40125A:                             ; CODE XREF: _main+1F↑j
.text:0040125A                 cmp     [ebp+var_C], 1440
.text:00401261                 jge     short loc_4012AF
.text:00401263                 mov     ecx, [ebp+var_C]
.text:00401266                 push    ecx
.text:00401267                 call    sub_401040
.text:0040126C                 add     esp, 4
.text:0040126F                 mov     [ebp+var_8], al
.text:00401272                 movsx   edx, [ebp+var_8]
.text:00401276                 test    edx, edx
.text:00401278                 jnz     short loc_40127E
.text:0040127A                 xor     eax, eax
.text:0040127C                 jmp     short loc_4012B1
.text:0040127E ; ---------------------------------------------------------------------------
.text:0040127E
.text:0040127E loc_40127E:                             ; CODE XREF: _main+48↑j
.text:0040127E                 movsx   eax, [ebp+var_8]
.text:00401282                 push    eax
.text:00401283                 push    offset aSuccessParsedC ; "Success: Parsed command is %c\
.text:00401288                 call    sub_4012B5
.text:0040128D                 add     esp, 8
.text:00401290                 mov     ecx, [ebp+argv]
.text:00401293                 mov     edx, [ecx]
.text:00401295                 push    edx             ; lpExistingFileName
.text:00401296                 mov     al, [ebp+var_8]
.text:00401299                 push    eax             ; char
.text:0040129A                 call    sub_401150
.text:0040129F                 add     esp, 8
.text:004012A2                 push    0EA60h          ; dwMilliseconds
.text:004012A7                 call    ds:Sleep
.text:004012AD                 jmp     short loc_401251
.text:004012AF ; ---------------------------------------------------------------------------
.text:004012AF
.text:004012AF loc_4012AF:                             ; CODE XREF: _main+31↑j
.text:004012AF                 xor     eax, eax
.text:004012B1
.text:004012B1 loc_4012B1:                             ; CODE XREF: _main+16↑j
.text:004012B1                                         ; _main+4C↑j
.text:004012B1                 mov     esp, ebp
.text:004012B3                 pop     ebp
.text:004012B4                 retn
.text:004012B4 _main           endp
.text:004012B4
```

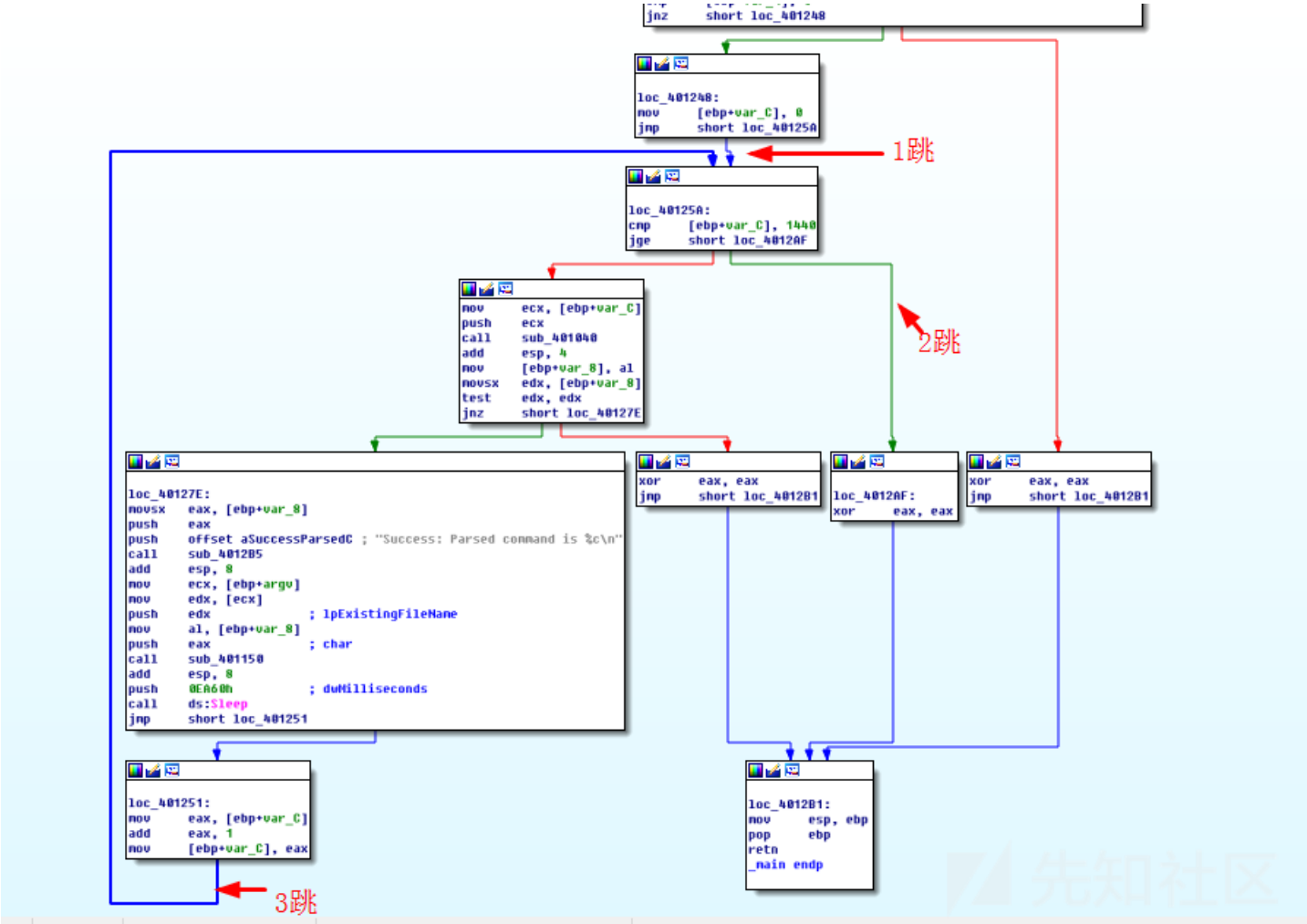
for循环

1. 初始化

2. 判断条件不成立后的跳出指令

3. 跳转到递增或递减处的语句块

可以大致将for语句分为3个语句块，初始化语句块、循环体（条件成立）、递增或递减语句块



这里用IDA视图来比较直观的观察for循环流程。

可以从里面的语句看到这里如果条件成立会循环1440次，并且每次还要睡眠1分钟，即这个程序在这里需要运行24小时

```
401040 hFile           = dword ptr -30h
401040 hInternet       = dword ptr -2Ch
401040 szAgent         = byte ptr -28h
401040 dwNumberOfBytesRead= dword ptr -8
401040 var_4           = dword ptr -4
401040 arg_0           = dword ptr  8
401040
401040                 push    ebp
401041                 mov     ebp, esp
401043                 sub     esp, 230h
401049                 mov     eax, [ebp+arg_0]
40104C                 push    eax
40104D                 push    offset aInternetExplor ; "Internet Explorer 7.50/pma%d"
401052                 lea     ecx, [ebp+szAgent]
401055                 push    ecx             ; char *
401056                 call    _sprintf
```

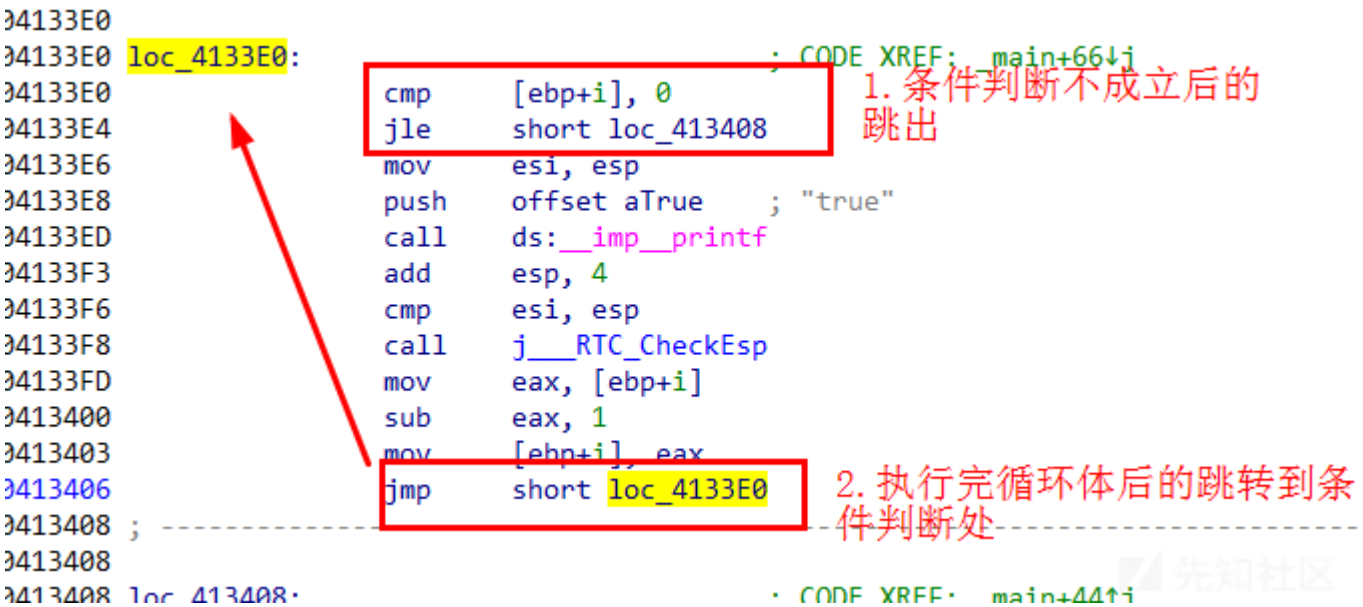
另外一个不同的地方就是这里会将循环次数传进这个函数，并且附加到这个代理字符串后面来访问网页文件，方便远程服务器知道大概的程序运行时间。别的功能都和上一个

while循环补充

while循环主要有：条件判断，条件成立后的循环体

所以while循环的汇编代码中只有两个跳转：

- 1.条件判断失败后跳出循环体
- 2.条件成立并执行完循环体后直接跳转到条件判断处继续循环



从图中可以看出while流程就是一个头部为判断条件尾部为直接跳转指令的语句块，因为只有两个跳转所以程序执行速度比使用for循环快

do-while流程补充

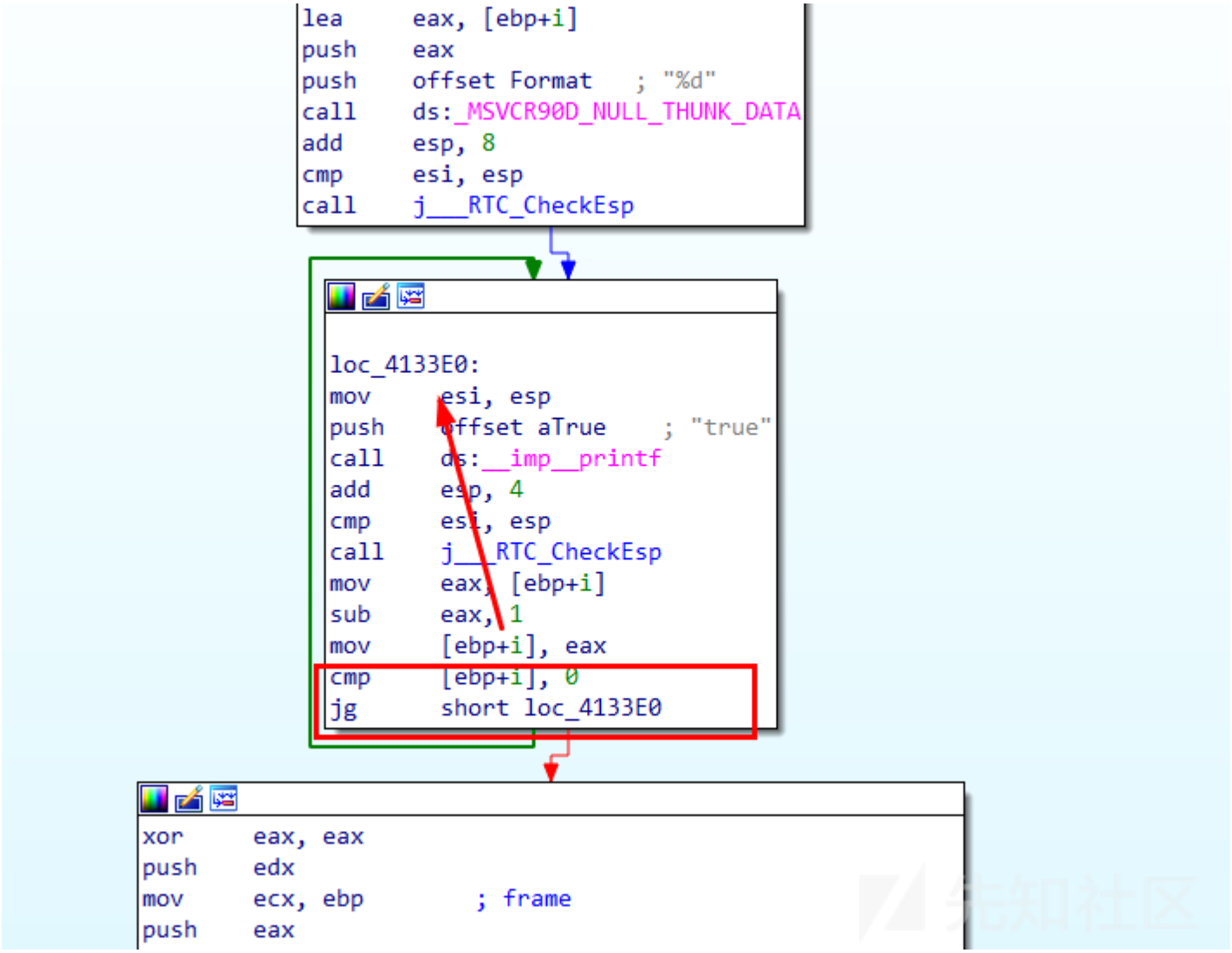
源代码:

```
#include <stdio.h>

void main()
{
   int i = 0;
   scanf("%d", &i);
   do
   {
       printf("true");
       i--;
   }
   while(i>0);
}
```

这个do-while流程和while的区别只是将头部的条件判断放到了尾部，所以头部的条件跳转和尾部的直接跳转融合成了一个条件跳转。

只有一个跳转的do-while流程比while流程更快



上一篇：Hybrid Android协议加... 下一篇：v8-Math.expm1-OOB...

1. 0 条回复
   - 动动手指，沙发就是你的了！