Sheridan College

Course	**INFO43921** **Malicious Code Design and Defense**
Activity Title	**Advanced Static/Dynamic Analysis (II)**
Student Name	Jackson Yuan
Lab performed on (Date):	April 14, 2023

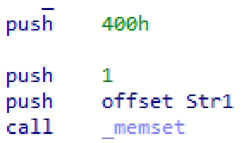
Objectives

- To be able to perform advance dynamic analysis with OllyDbg or IDA pro
- Able to set breakpoints and step through, over and into assembly code
- Able to read and follow assembly code register and be able to understand what is happening

Sample #1

Output#1: Purpose of the Malware

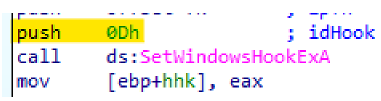
The purpose of this malware is to install a low-level keyboard hook that will monitor user keystrokes. The evidence for this is that it calls "_memset" which will initialize space and the specific location and the value to be set [1] as shown in Figure #1. Afterwards, it then installs a hook procedure to monitor the system for certain types of events by invoking the "SetWindowsHookExA" function [2]. The suspicion part of this code is when the hex code of "0D" is pass in which, if you were to convert it back to decimal you will get a value of "13". According to the documentations, a decimal value of 13 indicates that it installs a hook procedure that monitors low-level keyboard input events [2] as shown in Figure #2.



```
push    400h

push    1
push    offset Str1
call    _memset
```

Figure #1: Pushes in 400 bytes for the memory block size, pushes a value of 1 and sets a pointer with the offset

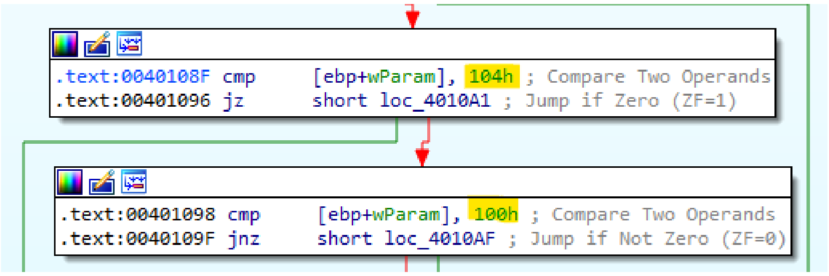


```
push    0Dh             ; idHook
call    ds:SetWindowsHookExA
mov     [ebp+hhk], eax
```

Figure #2: Hex value of 0D (13 in decimal) is pass into the parameters

Output#2: How Malware Injects Itself

The malware injects itself by using a technique called hook injection. Hook injection is a way to load malware that takes advantage of Windows hooks, which are used to intercept messages destined for applications [3]. In our case, this technique is used as a key logger to track user keystrokes. The keystroke is captured by invoking "WH_KEYBOARD_LL" which is a low-level hook as shown in Figure #2 [3] as stated above. Then it calls a the "offset fn" label in which is the area to actually check the keystroke message in the message queue of the window with the input focus [4] as shown in Figure #3; 104h and 100h are WM_KEYDOWN and WM_SYSKEYUP and these are doing the checking if a keypress or system keys has been released [4], [5].



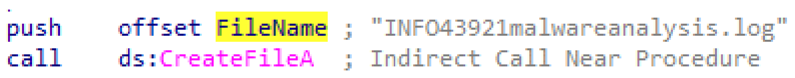
```
.text:0040108F cmp     [ebp+wParam], 104h ; Compare Two Operands
.text:00401096 jz      short loc_4010A1 ; Jump if Zero (ZF=1)
```

```
.text:00401098 cmp     [ebp+wParam], 100h ; Compare Two Operands
.text:0040109F jnz     short loc_4010AF ; Jump if Not Zero (ZF=0)
```

Figure #3: the "offset fn" label

Output#3: File Creation

It creates a .log file as Indicated by the figure below named "INFO43921malwareanalysis.log"



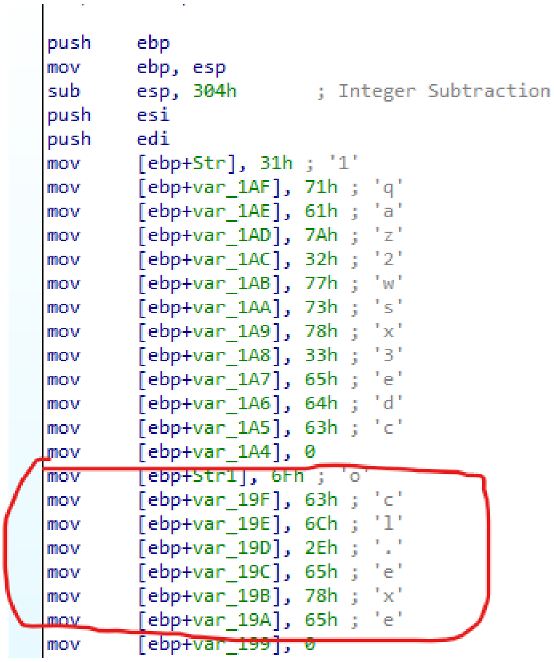
```
push    offset FileName ; "INFO43921malwareanalysis.log"
call    ds:CreateFileA  ; Indirect Call Near Procedure
```

Figure #4

Sample #2

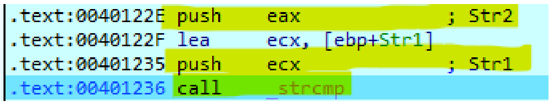
Output#1: Observe Live Payload

When the sample is executed, nothing is executed. When further inspecting the assembly code, we find that “ocl.exe” is hardcoded in as shown in Figure #5. This code can only be run if you change the filename to “ocl.exe” and it is shown running in Figure #6. Moreover, you can observer that there is a string comparison between the two strings within eax register and ecx registers in Figure #5.1. This is evident when looking at Figure #5.2 and Figure #5.3 as you can see that both contains the strings being compared. This basically checks if the final code only contains “ocl.exe”. If the string doesn’t work out to that then it will not execute.



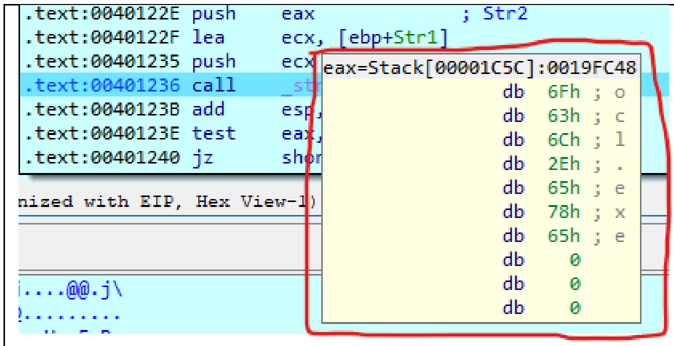
```
push    ebp
mov     ebp, esp
sub     esp, 304h       ; Integer Subtraction
push    esi
push    edi
mov     [ebp+Str], 31h ; '1'
mov     [ebp+var_1AF], 71h ; 'q'
mov     [ebp+var_1AE], 61h ; 'a'
mov     [ebp+var_1AD], 7Ah ; 'z'
mov     [ebp+var_1AC], 32h ; '2'
mov     [ebp+var_1AB], 77h ; 'w'
mov     [ebp+var_1AA], 73h ; 's'
mov     [ebp+var_1A9], 78h ; 'x'
mov     [ebp+var_1A8], 33h ; '3'
mov     [ebp+var_1A7], 65h ; 'e'
mov     [ebp+var_1A6], 64h ; 'd'
mov     [ebp+var_1A5], 63h ; 'c'
mov     [ebp+var_1A4], 0
mov     [ebp+Str1], 6Fh ; 'o'
mov     [ebp+var_19F], 63h ; 'c'
mov     [ebp+var_19E], 6Ch ; 'l'
mov     [ebp+var_19D], 2Eh ; '.'
mov     [ebp+var_19C], 65h ; 'e'
mov     [ebp+var_19B], 78h ; 'x'
mov     [ebp+var_19A], 65h ; 'e'
mov     [ebp+var_199], 0
```

Figure #5: “ocl.exe”



```
.text:0040122E push     eax             ; Str2
.text:0040122F lea      ecx, [ebp+Str1]
.text:00401235 push     ecx             ; Str1
.text:00401236 call     _strcmp
```

Figure #5.1: Comparing two strings to see if it’s valid



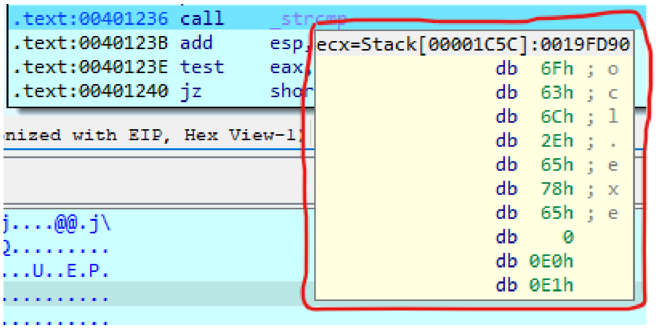
```
.text:0040122E push    eax             ; Str2
.text:0040122F lea     ecx, [ebp+Str1]
.text:00401235 push    ecx
.text:00401236 call    _st
.text:0040123B add     esp,
.text:0040123E test    eax,
.text:00401240 jz      shor

nized with EIP, Hex View-1)

i....@@.j\
?.........
```

```
eax=Stack[00001C5C]:0019FC48
                db  6Fh ; o
                db  63h ; c
                db  6Ch ; l
                db  2Eh ; .
                db  65h ; e
                db  78h ; x
                db  65h ; e
                db    0
                db    0
                db    0
```

Figure #5.2: String contained inside eax register



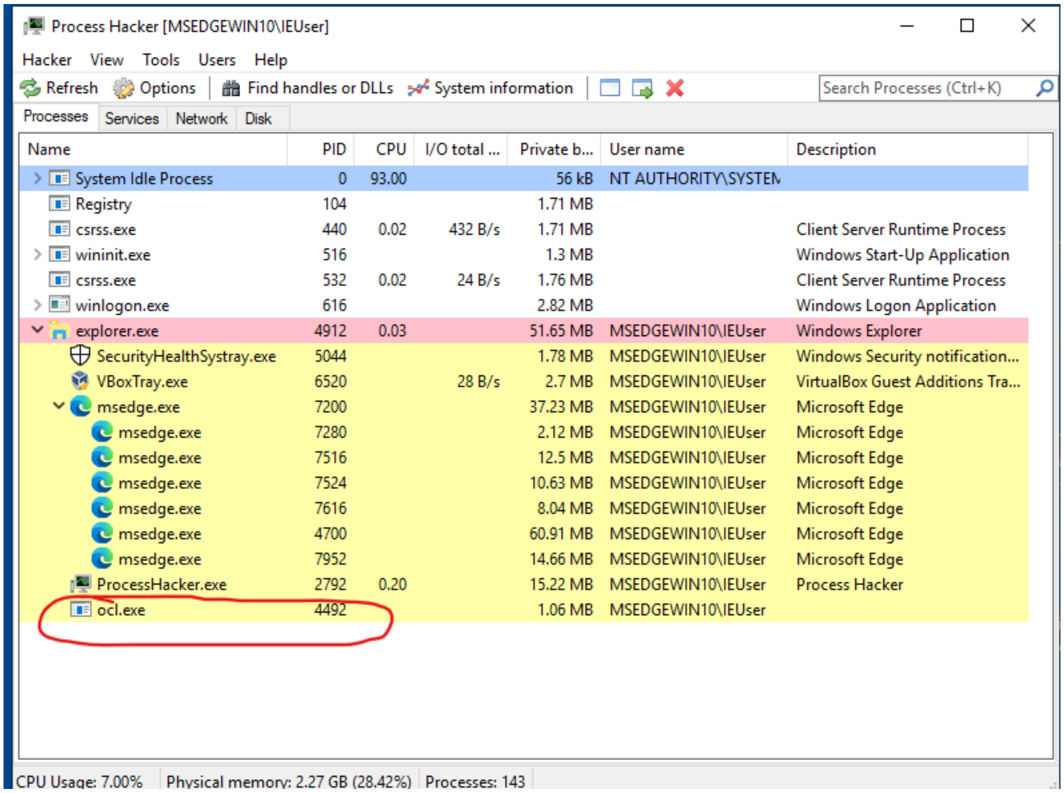
```
.text:00401236 call    _strcmp
.text:0040123B add     esp,
.text:0040123E test    eax,
.text:00401240 jz      shor

nized with EIP, Hex View-1)

j....@@.j\
?.........
...U..E.P.
..........
..........
```

```
ecx=Stack[00001C5C]:0019FD90
                db  6Fh ; o
                db  63h ; c
                db  6Ch ; l
                db  2Eh ; .
                db  65h ; e
                db  78h ; x
                db  65h ; e
                db    0
                db 0E0h
                db 0E1h
```

Figure #5.3: String contained inside ecx register



Process Hacker [MSEDGEWIN10\IEUser]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Processes (Ctrl+K)
Processes Services Network Disk

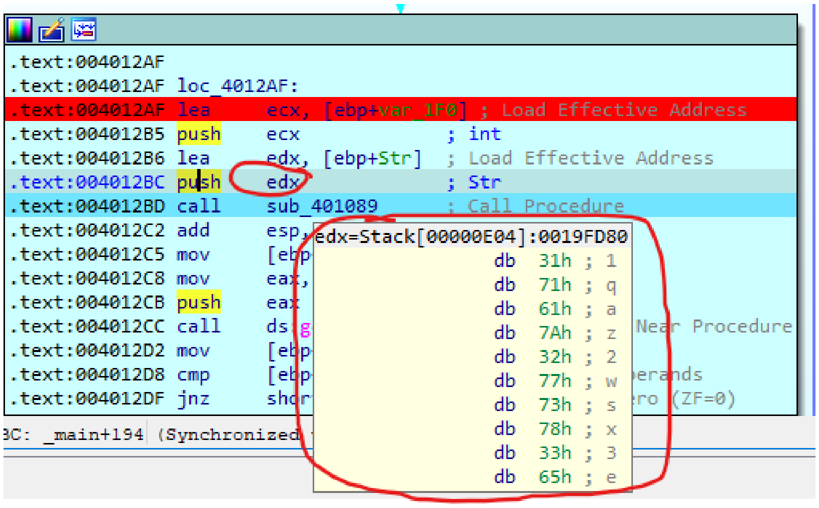
Name	PID	CPU	I/O total ...	Private b...	User name	Description
System Idle Process	0	93.00		56 kB	NT AUTHORITY\SYSTEM	
Registry	104			1.71 MB		
csrss.exe	440	0.02	432 B/s	1.71 MB		Client Server Runtime Process
wininit.exe	516			1.3 MB		Windows Start-Up Application
csrss.exe	532	0.02	24 B/s	1.76 MB		Client Server Runtime Process
winlogon.exe	616			2.82 MB		Windows Logon Application
explorer.exe	4912	0.03		51.65 MB	MSEDGEWIN10\IEUser	Windows Explorer
SecurityHealthSystray.exe	5044			1.78 MB	MSEDGEWIN10\IEUser	Windows Security notification...
VBoxTray.exe	6520		28 B/s	2.7 MB	MSEDGEWIN10\IEUser	VirtualBox Guest Additions Tra...
msedge.exe	7200			37.23 MB	MSEDGEWIN10\IEUser	Microsoft Edge
msedge.exe	7280			2.12 MB	MSEDGEWIN10\IEUser	Microsoft Edge
msedge.exe	7516			12.5 MB	MSEDGEWIN10\IEUser	Microsoft Edge
msedge.exe	7524			10.63 MB	MSEDGEWIN10\IEUser	Microsoft Edge
msedge.exe	7616			8.04 MB	MSEDGEWIN10\IEUser	Microsoft Edge
msedge.exe	4700			60.91 MB	MSEDGEWIN10\IEUser	Microsoft Edge
msedge.exe	7952			14.66 MB	MSEDGEWIN10\IEUser	Microsoft Edge
ProcessHacker.exe	2792	0.20		15.22 MB	MSEDGEWIN10\IEUser	Process Hacker
ocl.exe	4492			1.06 MB	MSEDGEWIN10\IEUser	

CPU Usage: 7.00% | Physical memory: 2.27 GB (28.42%) | Processes: 143

Figure #6: The sample finally executes when it has the name “ocl.exe”

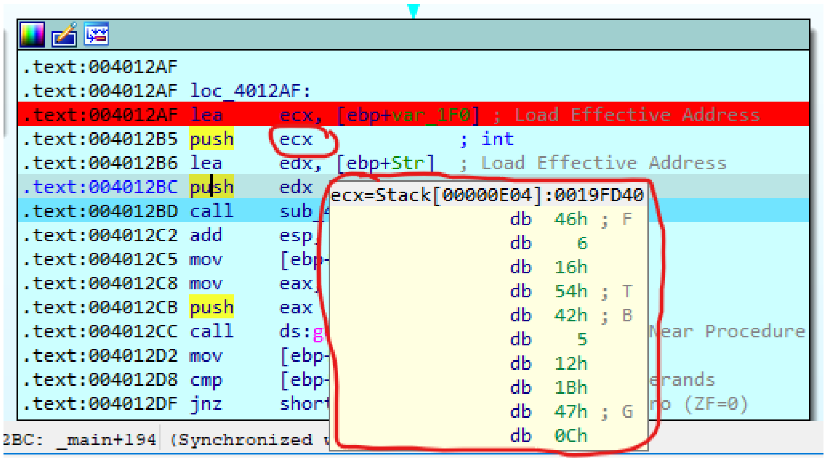
Output#2: String Observation

After converting the hex values to ascii, I get the following string “1qa2wsx3edc”. The two arguments that are being passed into subroutine 0x00401089 is the string “1qa2wsx3e” in the edx register and the effective address of var_1F0 which is contained inside register ecx.



```
.text:004012AF
.text:004012AF loc_4012AF:
.text:004012AF lea     ecx, [ebp+var_1F0] ; Load Effective Address
.text:004012B5 push    ecx               ; int
.text:004012B6 lea     edx, [ebp+Str]    ; Load Effective Address
.text:004012BC push    edx               ; Str
.text:004012BD call    sub_401089        ; Call Procedure
.text:004012C2 add     esp,
.text:004012C5 mov     [ebp
.text:004012C8 mov     eax,
.text:004012CB push    eax
.text:004012CC call    ds:g                       Near Procedure
.text:004012D2 mov     [ebp
.text:004012D8 cmp     [ebp                      erands
.text:004012DF jnz     shor                      ro (ZF=0)
3C: _main+194 (Synchronized
```

```
edx=Stack[00000E04]:0019FD80
                   db  31h ; 1
                   db  71h ; q
                   db  61h ; a
                   db  7Ah ; z
                   db  32h ; 2
                   db  77h ; w
                   db  73h ; s
                   db  78h ; x
                   db  33h ; 3
                   db  65h ; e
```

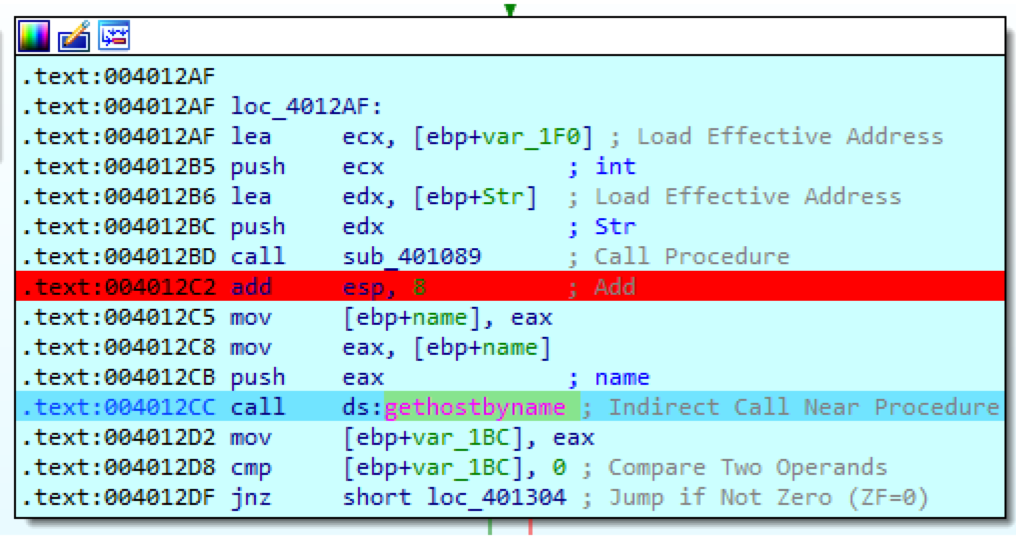


```
.text:004012AF
.text:004012AF loc_4012AF:
.text:004012AF lea     ecx, [ebp+var_1F0] ; Load Effective Address
.text:004012B5 push    ecx               ; int
.text:004012B6 lea     edx, [ebp+Str]    ; Load Effective Address
.text:004012BC push    edx
.text:004012BD call    sub_4
.text:004012C2 add     esp,
.text:004012C5 mov     [ebp-
.text:004012C8 mov     eax,
.text:004012CB push    eax
.text:004012CC call    ds:g                       Near Procedure
.text:004012D2 mov     [ebp-
.text:004012D8 cmp     [ebp-                      erands
.text:004012DF jnz     short                      ro (ZF=0)
2BC: _main+194 (Synchronized w
```

```
ecx=Stack[00000E04]:0019FD40
                   db  46h ; F
                   db    6
                   db  16h
                   db  54h ; T
                   db  42h ; B
                   db    5
                   db  12h
                   db  1Bh
                   db  47h ; G
                   db  0Ch
```

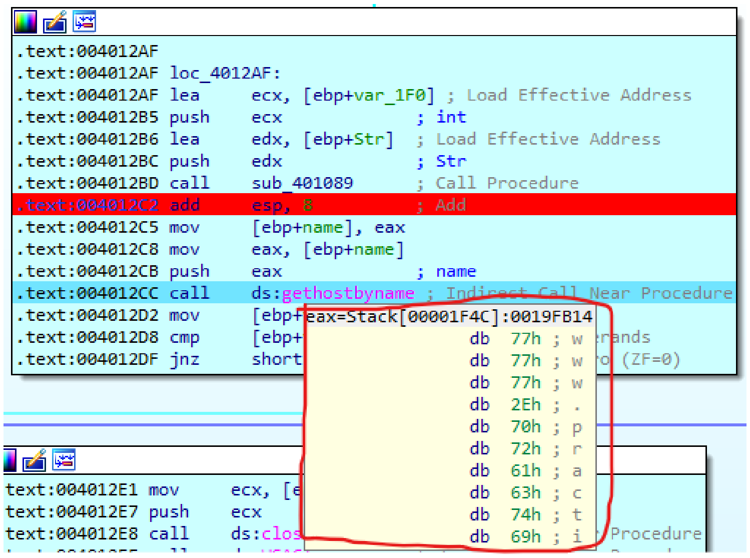
Output#3: Domain Name

The domain name is www.practicalmalwareanalysis.com. The reason how I came to this answer is I looked for any suspicious network API calls, which lead me to “gethostbyname”. Looking at gethostbyname’s API, we find that it has a parameter that is called “name” which contains the actual IP address [6], [7]. In order to see what it passes I had to add a break point a few addresses before the gethostbyname API is called as shown in Figure #7. Once ran, we see that the name is contained within the eax register address 0019FB14 as shown in Figure #8 & #9.



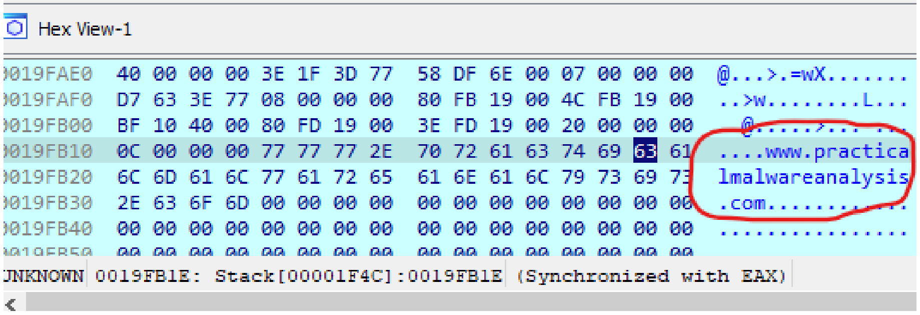
```
.text:004012AF
.text:004012AF loc_4012AF:
.text:004012AF lea     ecx, [ebp+var_1F0] ; Load Effective Address
.text:004012B5 push    ecx                ; int
.text:004012B6 lea     edx, [ebp+Str]     ; Load Effective Address
.text:004012BC push    edx                ; Str
.text:004012BD call    sub_401089         ; Call Procedure
.text:004012C2 add     esp, 8             ; Add
.text:004012C5 mov     [ebp+name], eax
.text:004012C8 mov     eax, [ebp+name]
.text:004012CB push    eax                ; name
.text:004012CC call    ds:gethostbyname ; Indirect Call Near Procedure
.text:004012D2 mov     [ebp+var_1BC], eax
.text:004012D8 cmp     [ebp+var_1BC], 0 ; Compare Two Operands
.text:004012DF jnz     short loc_401304 ; Jump if Not Zero (ZF=0)
```

Figure #7: Breakpoint added at .text:004012C2



```
.text:004012AF
.text:004012AF loc_4012AF:
.text:004012AF lea     ecx, [ebp+var_1F0] ; Load Effective Address
.text:004012B5 push    ecx                ; int
.text:004012B6 lea     edx, [ebp+Str]     ; Load Effective Address
.text:004012BC push    edx                ; Str
.text:004012BD call    sub_401089         ; Call Procedure
.text:004012C2 add     esp, 8             ; Add
.text:004012C5 mov     [ebp+name], eax
.text:004012C8 mov     eax, [ebp+name]
.text:004012CB push    eax                ; name
.text:004012CC call    ds:gethostbyname ; Indirect Call Near Procedure
.text:004012D2 mov     [ebp+eax=Stack[00001F4C]:0019FB14
.text:004012D8 cmp     [ebp+                db   77h ; w   erands
.text:004012DF jnz     short                db   77h ; w   o (ZF=0)
                                            db   77h ; w
                                            db   2Eh ; .
                                            db   70h ; p
                                            db   72h ; r
text:004012E1 mov     ecx, [e               db   61h ; a
text:004012E7 push    ecx                   db   63h ; c
text:004012E8 call    ds:clos               db   74h ; t
                                            db   69h ; i   Procedure
```

Figure #8: Locating address 0019FB14



```
0019FAE0  40 00 00 00 3E 1F 3D 77  58 DF 6E 00 07 00 00 00  @...>.=wX.......
0019FAF0  D7 63 3E 77 08 00 00 00  80 FB 19 00 4C FB 19 00  ..>w........L...
0019FB00  BF 10 40 00 80 FD 19 00  3E FD 19 00 20 00 00 00  ..@.....>.......
0019FB10  0C 00 00 00 77 77 77 2E  70 72 61 63 74 69 63 61  ....www.practica
0019FB20  6C 6D 61 6C 77 61 72 65  61 6E 61 6C 79 73 69 73  lmalwareanalysis
0019FB30  2E 63 6F 6D 00 00 00 00  00 00 00 00 00 00 00 00  .com............
0019FB40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0019FB50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
UNKNOWN 0019FB1E: Stack[00001F4C]:0019FB1E (Synchronized with EAX)
```

Figure #9: Hex view of the information contained within address 0019FB14.

References

[1] "C Library Function - memset()," *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_memset.htm. [Accessed: 06-Apr-2023].

[2] "Setwindowshookexa function (winuser.h) - win32 apps," *Win32 apps | Microsoft Learn*, 08-Feb-2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa. [Accessed: 06-Apr-2023].

[3] M. Sikorski and A. Honig, "Practical malware analysis," *O'Reilly Online Learning*. [Online]. Available: https://learning.oreilly.com/library/view/practical-malware-analysis/9781593272906/ch13s04.html. [Accessed: 06-Apr-2023].

[4] C. Petzold, "Programming windows®, Fifth Edition," *O'Reilly Online Learning*. [Online]. Available: https://learning.oreilly.com/library/view/programming-windows-r-fifth/9780735642225/ch06s02.html. [Accessed: 06-Apr-2023].

[5] "List of windows messages," *List Of Windows Messages - WineHQ Wiki*. [Online]. Available: https://wiki.winehq.org/List_Of_Windows_Messages. [Accessed: 06-Apr-2023].

[6] "Gethostbyname function (winsock2.h) - win32 apps," *Win32 apps | Microsoft Learn*, 21-Sep-2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-gethostbyname. [Accessed: 07-Apr-2023].

[7] *Gethostbyname subroutine*, 24-Mar-2023. [Online]. Available: https://www.ibm.com/docs/en/aix/7.1?topic=g-gethostbyname-subroutine. [Accessed: 07-Apr-2023].