Sheridan College

Course	INFO43921 Malicious Code Design and Defense
Activity Title	**Advanced Static/Dynamic Analysis**
Student Name	Jackson Yuan
Lab performed on (Date):	March 31, 2023

Objectives

- Understand the usage of OllyDbg & IDA Pro
- Inspect and devise malicious code by inspecting assembly language
- Understand how malware achieve persistence through analysis of assembly code

Output Section for Sample #1

Output #1: Sample Persistence

This sample achieves persistence by creating a service called “Newservice” as shown in Figure #1. By definition, operating systems have provisions to start processes automatically when system boots [8] thus by creating this service, the malware will stay persistence. Additionally, when there exist any scheduled tasks created, then it is also an indication of malware trying to be persistent which is what we can see in Figure #5, this is mainly to periodically execute the malware.

```
.text:004010AF 6A 02                    push    2               ; dwDesiredAccess
.text:004010B1 68 3C 50 40 00           push    offset DisplayName ; "Newservice"
.text:004010B6 68 3C 50 40 00           push    offset DisplayName ; "Newservice"
.text:004010BB 56                       push    esi             ; hSCManager
.text:004010BC FF 15 00 40 40 00        call    ds:CreateServiceA ; Indirect Call Near Procedure
.text:004010C2 33 D2                    xor     edx, edx        ; Logical Exclusive OR
.text:004010C4 8D 44 24 14              lea     eax, [esp+404h+FileTime] ; Specifies a date and time
```

Figure #1 above: A new service is created and is named “Newservice”.

Output #2: Mutex Definition & Mutex in Sample

By definition, a mutex is a program object that ensures only one instance of a program get access to resources. In the context of malware, the malware author ensures that their malicious program does not infect the system multiple times [1]. In our sample, the reason

why it uses a mutex is because it is trying to enforce a single instance of itself due to the fact that it is using resources to consistently access the URL http://www.INFO43921_Sample_01.com shown in Figure #4.

```
esi
offset Name       ; "S43921"
0                 ; bInitialOwner
0                 ; lpMutexAttributes
ds:CreateMutexA   ; Indirect Call Near Procedure
3                 ; dwDesiredAccess
0                 ; lpDatabaseName
0                 ; lpMachineName
```

Figure #2 above: Creating a mutex with the name "S43921".

```
405048 ; CHAR Name[]
405048 Name            db 'S43921',0          ; DATA XREF: sub_401040+6↑o
405048                                        ; sub_401040+25↑o
40504F                 align 10h
```

Figure #3 aboce: Notice only one instance of the mutex name is created; is reference twice.

```
push    offset szAgent  ; "Internet Explorer 8.0"
call    ds:InternetOpenA ; Indirect Call Near Procedure
mov     edi, ds:InternetOpenUrlA
mov     esi, eax

                        ; CODE XREF: StartAddress+30↓j
push    0               ; dwContext
push    80000000h       ; dwFlags
push    0               ; dwHeadersLength
push    0               ; lpszHeaders
push    offset szUrl    ; "http://www.INFO43921_Sample_01.com"
push    esi             ; hInternet
call    edi ; InternetOpenUrlA ; Indirect Call Near Procedure
jmp     short loc_40116D ; Jump
```

Figure #4: URL being loaded and internet explorer 8.0 being used.

Output #3: Purpose of Sample & Signature Suggestions

The purpose of this sample is to continuously access a weblink http://www.INFO43921_Sample_01.com by invoking the function "InternetOpenA" which initializes an application's use of the WinINet functions and connected to Internet Explorer 8.0 [2] with the parameters szAgent. Then finally the sample opens the URL by calling the InternetOpenUrlA function [3] as all of this is shown in Figure #4. Nevertheless, the underlying mechanism is that the malware creates a timer by invoking the SystemTimeToFileTime function in which it converts a system time to file time format shown in Figure #5 [4]. From there, it calls the SetWaitableTimer function which will activate the specified waitable timer shown in Figure #5. When the due time arrives, the timer is signaled

and the thread that set the timer calls the optional completion routine [5]. This is evident when we see the next function call WaitForSingleObject [6] which waits until the SetWaitableTimer object is in the signaled state or the time-out interval elapses shown in Figure #5. Once the timer has decremented to zero, it will suspend the execution for a set period of time by invoking the Sleep function [8]. Once this occurs, it will start back from the beginning again by calling subroutine shown in Figure #6 & #7.

My suggestion for this specific malware would be first, a good host-based signature will be the malware mutexes. In our case, the mutex name is "S43921" as this is hard coded in to avoid further conflicts as per the definition of a mutex aforementioned. Finally, in terms of a network-based signature, we can detect the by URL; http://www.INFO43921_Sample_01.com which, as aforementioned the malware opens the link to this website.

```
[esp+40Ch+SystemTime.wYear], 834h
ds:SystemTimeToFileTime ; Converts a system time to file time format. System time is based on Co
0               ; lpTimerName
0               ; bManualReset - If this parameter is TRUE, the timer is a manual-reset notifica
0               ; lpTimerAttributes
ds:CreateWaitableTimerA ; Indirect Call Near Procedure
0               ; fResume
0               ; lpArgToCompletionRoutine
0               ; pfnCompletionRoutine
edx, [esp+410h+FileTime] ; Load Effective Address
esi, eax
0               ; lPeriod
edx             ; lpDueTime
esi             ; hTimer
ds:SetWaitableTimer ; Activates the specified waitable timer. When the due time arrives, the tim
0FFFFFFFFh      ; dwMilliseconds
esi             ; hHandle
ds:WaitForSingleObject ; Watches for the setwaitable timer above to be true
eax, eax        ; Logical Compare
short loc_40113B ; Jump if Not Zero (ZF=0)
```

Figure #5: Explanation of code.

```
loc_40113B:             ; dwMilliseconds
push    0FFFFFFFFh
call    ds:Sleep        ; Suspends the execution of the current thread until the time-out interval elapses
xor     eax, eax        ; Logical Exclusive OR
pop     esi
add     esp, 400h       ; Add
retn                    ; Return Near from Procedure
sub_401040 endp
```

Figure #6: Sleep function.

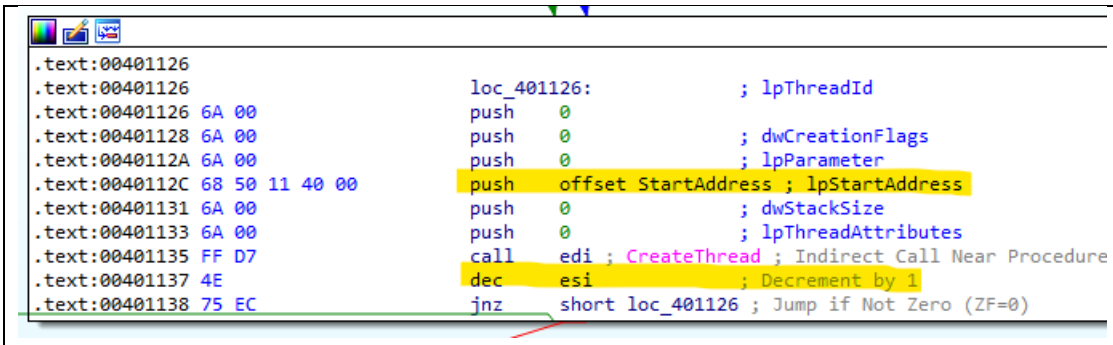

Figure #7: Loop that tries to open website.

Output Section for Sample #2

Output #4: Sample Persistence

This sample does not have any persistence as by using the definition, a malware achieves persistence by having access to the system even after restarts systems across restarts, changed credentials, and other interruptions that could cut off their access [8]. Knowing this fact, sample #2 does none of the aforementioned things as explained in output #5 on how this sample behaves.

Output #5: Purpose of Sample

The overall purpose of this sample is that it is most likely trying to communicate externally with http://www.INFO43921Sample0102.com/xy.html. The evidence of this is that it first, initialize the COM library on a particular thread as by definition, COM is a binary-interface standard that enables communication between objects and their clients, regardless of the programming language they are written in Figure #8 [9]. The next function call is to create the object itself by invoking CoCreateInstance which creates an instance of a COM object; only one object on the local system [10] as shown in Figure #9. A variant structure was then created to pass data between different components in which this will pass a string with the URL "http://www.INFO43921Sample0102.com/xy.html" to SysAllocString. It then calls a function at the address in the edx register (this is from "call dword ptr [edx+2Ch]") with the URL string as seen in Figure #10. It then deallocates the string by calling the SysFreeString method [11] as shown in Figure #10. Finally, it uninitialized the COM library by calling OleUninitialize [12] as shown in Figure #11.

```
00401000
00401000 sub     esp, 24h        ;
00401000                         ;
00401000                         ;
00401003 push    0               ;
00401005 call    ds:OleInitialize ;
0040100B test    eax, eax        ;
0040100D jl      short loc_401085 ;
```

Figure #8: Initialize of COM library.

```
00401013 push    eax             ; p
00401014 push    offset riid     ; r
00401019 push    4               ; d
0040101B push    0               ; p
0040101D push    offset rclsid   ; r
00401022 call    ds:CoCreateInstance
00401028 mov     eax, [esp+24h+ppv]
0040102C test    eax, eax        ; L
0040102E jz      short loc_40107F ;
```

Figure #9: Creates an instance of a COM object.

```
00401073 push    eax
00401074 call    dword ptr [edx+2Ch] ; Indirect Call Near Procedure
00401077 push    esi             ; bstrString: The previously allocate
00401078 call    ds:SysFreeString ; Deallocates (Frees memory) a strir
0040107E pop     esi
```

Figure #10: Calls a method of the COM object

```
0040107F
0040107F loc_40107F:             ; this
0040107F call    ds:OleUninitialize ;
0040107F                         ; OleU
```

Figure #11: Uninitialized the COM library.

Output#6: How and When This Sample Finish Its Execution

As described in Output #5, this sample will finish its execution when it has ran through all the process as aforementioned it will free up memory allocated for the URL string by invoking SysFreeString (as shown in Figure #11) then closes the COM library on the apartment, releases any class factories, other COM objects, or servers held by the apartment, disables RPC on the apartment, and frees any resources the apartment maintains [11]. Finally it will return back to the caller where the main function ends as shown in Figure #12 below.

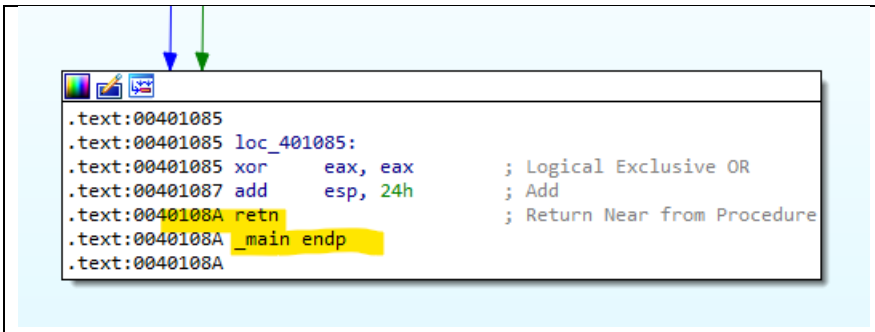

Figure #12: return to caller and end of main function.

References

[1] J. Blasco, “Malware: Exploring mutex objects,” *AT&T Cybersecurity*, 28-Dec-2009. [Online]. Available: https://cybersecurity.att.com/blogs/labs-research/malware-exploring-mutex-objects. [Accessed: 26-Mar-2023].

[2] QuinnRadich, “Internetopena function (wininet.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 02-Aug-2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetopena. [Accessed: 28-Mar-2023].

[3] QuinnRadich, “Internetopenurla function (wininet.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 08-Feb-2023. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetopenurla. [Accessed: 28-Mar-2023].

[4] Karl-Bridge-Microsoft, “SystemTimeToFileTime function (timezoneapi.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 10-Dec-2021. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/timezoneapi/nf-timezoneapi-systemtimetofiletime. [Accessed: 28-Mar-2023].

[5] Karl-Bridge-Microsoft, “Setwaitabletimer function (synchapi.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 22-Sep-2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-setwaitabletimer. [Accessed: 28-Mar-2023].

[6] Karl-Bridge-Microsoft, “WaitForSingleObject function (synchapi.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 22-Sep-2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject. [Accessed: 28-Mar-2023].

[7] Karl-Bridge-Microsoft, “Sleep function (synchapi.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 22-Sep-2022. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-sleep. [Accessed: 28-Mar-2023].

[8] A. Hassan, “Week 6.1 - Persistence,” *Login - sheridan learning and teaching environment*, 14-Feb-2023. [Online]. Available: https://slate.sheridancollege.ca/d2l/le/content/1058873/viewContent/13865729/View. [Accessed: 28-Mar-2023].

[9] Stevewhims, “Oleinitialize function (ole2.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 10-Dec-2021. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/ole2/nf-ole2-oleinitialize. [Accessed: 30-Mar-2023].

[10] Stevewhims, “Cocreateinstance function (combaseapi.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 17-Oct-2021. [Online]. Available: https://learn.microsoft.com/en-

us/windows/win32/api/combaseapi/nf-combaseapi-cocreateinstance. [Accessed: 30-Mar-2023].

[11] TylerMSFT, “Allocating and releasing memory for a BSTR,” *Microsoft Learn*, 12-Jan-2022. [Online]. Available: https://learn.microsoft.com/en-us/cpp/atl-mfc-shared/allocating-and-releasing-memory-for-a-bstr?view=msvc-170. [Accessed: 30-Mar-2023].

[12] Stevewhims, “Oleuninitialize function (ole2.h) - win32 apps,” *Win32 apps | Microsoft Learn*, 28-Jun-2021. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/ole2/nf-ole2-oleuninitialize. [Accessed: 30-Mar-2023].