7th Lab - Windows malware

Rodrigo Castillo and Juan Esteban Murcia

September 21, 2020

1 Theorical Lab

1.1 Read the introduction to the Section "Analyzing Malicious Windows Programs" and explain why is it important to know the details of Windows OS (Windows API, user/kernel mode, execution of code outside a file)?

As a computer science student, is important to know details about Windows OS because is the most used operating system , as a forensics investigator, is important because most of the malware works for windows, somethimes, malware will use those libraries and services making the aknowledgment of those libraries really important for understanding and studyng malware and how it works.

1.2 Read the section "The Windows API" and identify which are the Windows API Types

.

types of windows api:

- WORD: is a unsigned 16bit value .
- DWORD: is a unsigned 32bit value .
- HANDLES: is a reference to an object , is not documented so it should only be managed by windows APIs.
- Long Pointer is a pointer to another type of variable.
- Callback represents a function that is going to be called by a windows API.

1.3 Read the section "File System Functions" and explain the difference between shared files and files accesible via namespaces

shared files are special files which paths looks like $//nameserver/share$ **or** $//?/nameserver/share$. the symbol tells the operating system to not parse the string, in order to access longer filenames. Files accesible via namespaces: namespaces can be understood as the number of a folder, each one store different type of objects. lowest namespace is called **NT** and it has access to all devices and all namespaces exist inside **NT**. An example of files found within the NT namespace can be seen in the following figure:

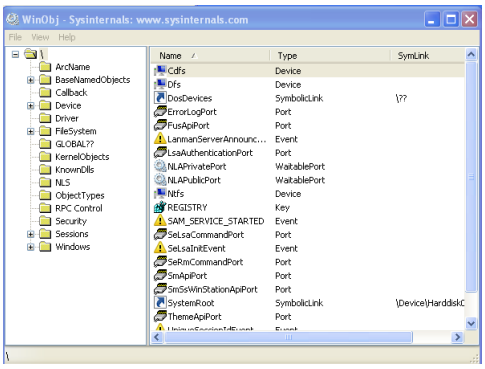

Figure 1: Namespace NT

1.4 Read the section "The Windows Registry" and identify the place where executables (.exe) that start up when user log in may be configured.

A program that needs to start when a user logs in can be configured using the Run subkey using the Autoruns tool, that is a free tool from Microsoft that list code and executables that will be executed on start up. In the following figure we can see the default programs that starts when the user logs in, and we can validate it is located in the Run subkey.

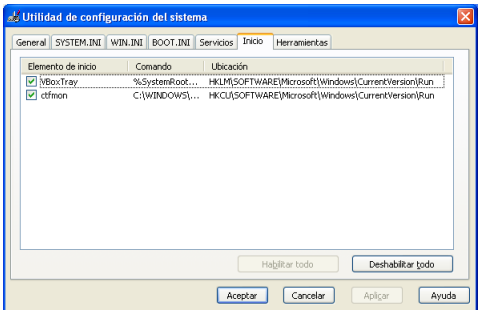

Figure 2: Startup programs

And here a screenshot of the Run subkey location using regedit:

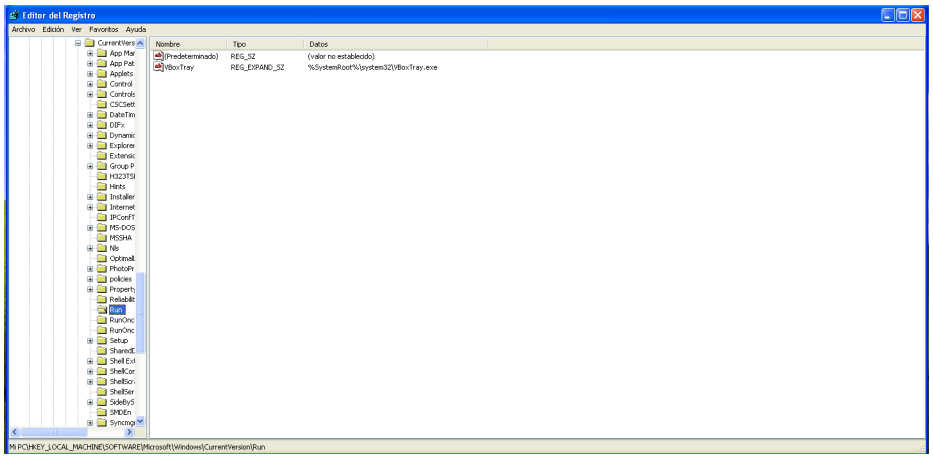

Figure 3: Run location

1.5 Read the section "Common Registry Functions" and explain the difference between RegOpenKeyEx, RegSetValueEx and RegGetValue

- RegOpenKeyEx
  Opens a registry for editing a query it contains a functions that allows to edit the registry without open it. Most of the programs use this one.
- RegSetValueEx
  It adds a value to a registry and modifies the data
- RegGetValue
  It returns the data found inside a registry.

2 Explain the following assembly code. What is the purpose?

In the first place it calls the function RegOpenKeyEx to open the Run subkey, then it start handling and processing what we identified as a string to finally putting it in the Run registry using the function RegSetValueEx.

2.1 Read the Section "The WinINet API" (Pag 178) and describe what the function InternetOpen does?

The WinInet API is a high level API which functions are stored in wininit.dll, if a program imports functions from this library, it's using high level functions with networking purposes. One of those functions is InternetOpen that initialize an internet conection. But there are also other functions such as InternetOpenURL and InternetReadFile.

2.2 Read the section "Services" (Pag 185) and explain which are the main functions to manage services?

Another way to execute aditional code is stalling a *service*, because windows allows to run tasks using services. To achive this we have the three following functions:

- OpenSCManager, it returns a handle to the service, all code that wants to use a service approach must call this function.
- CreateService, This function adds a new service to the service control man ager, this allows the caller to specify where it will be executed and when it's going to be called.
- StartService, It starts a new service, it is only used if such service is going to be configured manually.

2.3 Read the section "Interprocess Coordination with Mutexes" (Pag 184), understand the following code and explain why is important a mutex?

Mutex refers to global objects, that coordinates several proceses and execution threads, generally they are used to control access to shared resources, but they are also used in malware, because if two different threads needs to access the same space, a mutex can control this operations. The assembly code what is basically doing is trying to open a mutex using the three arguments pushed to the stack, we assume that if the operation was successfull, it jumps to another part of the execution, else it proceed to create a the mutex with those three arguments, we can interpret that as creating the execution of a malware if and oly if such malare is not already in execution.

2.4 What is a SYSTEMTIME structure?

The systemtime structure specifies date and time, using members like month, day, year, weekday, minute, second and millisecond, it is either in UTC or in local time.

2.5 What the SetWaitableTimer function does?

It activates a specified waitable timer, when the due time arrives, the timer is signaled and the thread of the timer calls an optional routine.

2.6 Read the section "Creating a Thread" (Pag 182), understand the following code and explain in detail how a thread is set?

CreateThread function creates a new thread of execution, so what we can see in the code is ho the malware is creating 2 different threads with several mlicious purposes, for example load libraries in parallel to avoid stoping the malware execution or to avoid exiting a while, etc.

3 Practical Lab

3.1 Identify in the "Strings" tab, some functions that may indicate that the malware configure a service (OpenSCManager, CreateService, etc).

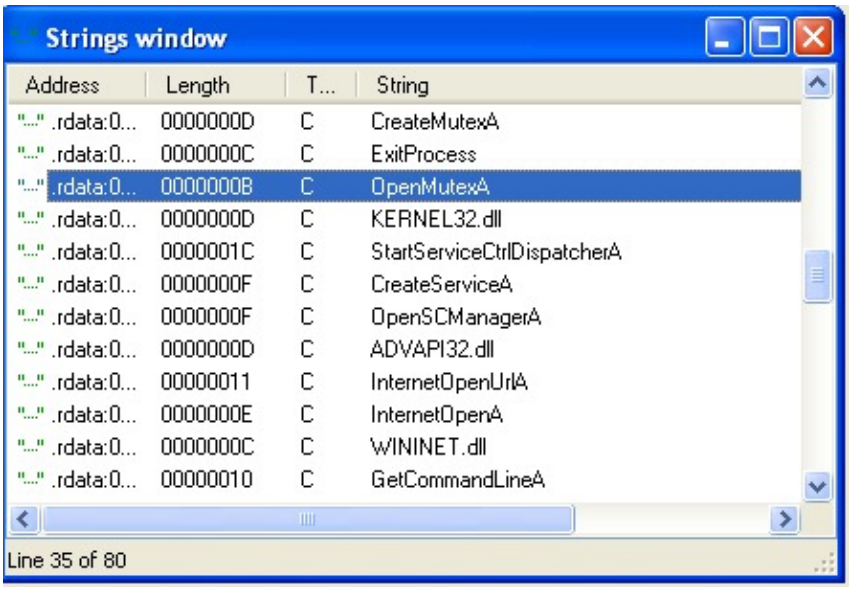

Figure 4: Strings related to services

here, we can see that the functions that we identified previously are in the source binary of the program, that mean that probably the program is consuming them. This make the previus section important to understand this kind of malware.

3.2 location of main

Using Ida we identified the location of main main fuction is locaten at $0x00401000$.

```
.text:00401000 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401000 _main           proc near               ; CODE XREF: start+AF↓p
.text:00401000
.text:00401000 ServiceStartTable= SERVICE_TABLE_ENTRYA ptr -10h
.text:00401000 var_8           = dword ptr -8
.text:00401000 var_4           = dword ptr -4
.text:00401000 argc            = dword ptr  4
.text:00401000 argv            = dword ptr  8
.text:00401000 envp            = dword ptr  0Ch
.text:00401000
.text:00401000                 sub     esp, 10h
.text:00401003                 lea     eax, [esp+10h+ServiceStartTable]
.text:00401007                 mov     [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ; "MalService"
.text:0040100F                 push    eax             ; lpServiceStartTable
.text:00401010                 mov     [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
.text:00401018                 mov     [esp+14h+var_8], 0
.text:00401020                 mov     [esp+14h+var_4], 0
.text:00401028                 call    ds:StartServiceCtrlDispatcherA
.text:0040102E                 push    0
.text:00401030                 push    0
.text:00401032                 call    sub_401040
.text:00401037                 add     esp, 18h
.text:0040103A                 retn
.text:0040103A _main           endp
```

Figure 5: location of main function

3.3 Review the code inside function sub_401040 and observe that inside there is also a call to the funcion $OpenMutexA$.

is calling a mutex function to check if malware is already created, if the malware is created, the function call ends, but if is not created yet, the mutex for the malware is created.

```
.text:00401040 sub_401040      proc near               ; CODE XREF: _main+32↑p
.text:00401040                                         ; DATA XREF: _main+10↑o
.text:00401040
.text:00401040 SystemTime      = SYSTEMTIME ptr -400h
.text:00401040 DueTime         = LARGE_INTEGER ptr -3F0h
.text:00401040 BinaryPathName  = byte ptr -3E8h
.text:00401040
.text:00401040                 sub     esp, 400h
.text:00401046                 push    offset Name     ; "HGL345"
.text:0040104B                 push    0               ; bInheritHandle
.text:0040104D                 push    1F0001h         ; dwDesiredAccess
.text:00401052                 call    ds:OpenMutexA
.text:00401058                 test    eax, eax
.text:0040105A                 jz      short loc_401064
.text:0040105C                 push    0               ; uExitCode
.text:0040105E                 call    ds:ExitProcess
.text:00401064
.text:00401064 loc_401064:                             ; CODE XREF: sub_401040+1A↑j
.text:00401064                 push    esi
.text:00401065                 push    offset Name     ; "HGL345"
.text:0040106A                 push    0               ; bInitialOwner
.text:0040106C                 push    0               ; lpMutexAttributes
.text:0040106E                 call    ds:CreateMutexA
.text:00401074                 push    3               ; dwDesiredAccess
.text:00401076                 push    0               ; lpDatabaseName
.text:00401078                 push    0               ; lpMachineName
.text:0040107A                 call    ds:OpenSCManagerA ; Establish a connection to the service
.text:0040107A                                         ; control manager on the specified computer
.text:0040107A                                         ; and opens the specified database
```

Figure 6: Mutex call

3.4 Analyze the code at the address 401064 and observe that at 40106E there is a call to the function ds:CreateMutexA. Then, there is also a call to functions ds:OpenSCManager and ds:GetModuleFileName. ds:GetModuleFileName gets the full pathname to the running executable or Loaded DLL.

Figure 7: service path

3.5 Observe that at address 004010C2 exitst a SYSTEMTIME... SystemTime.wYear is 834h. What does it mean

it means that is taking a variable in system date, and is modifying this date to year 2100.

4 Question:

4.1 Q13: How does astaroth.exe ensure that it continues running (achieves persistence) when the computer is restarted?

creating itself as a service, as we know in previus section , creating services will execute as autostart in windows when the sytem boots.

4.2 Q14: Why does astaroth.exe use a mutex?

creating itself as a service but only the first time or if the service was elimitated inorder to avoid creating the same service several times.

4.3 Q15: What is a good host-based signature to use for detecting astaroth.exe?

if there is a service that is wating untill year 2100 and it also importing mutex as a library is higly probable that this program is astaroth or similar malware. there are other several dll files that can be used to identify astaroth with more efficiency .

4.4 Q16: When will astaroth.exe finish executing?

in a day of the year 2100 .