Lab7
Xiaoyi Zhou

Lab7-01

1. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?

Before we analyze the program, we should get to know the Windows API called by the program.

In the area code of main function, we noticed an API calls StartServiceCtrlDispatcher (), the purpose of this function is to connects the main process to the service control manager. When the function is called, we can assume that the program will create a service and execute it in following seconds. Also the function has a very important parameter denoted as IpserviceStartTable, I do regard it as a table array because the structure contains one value for each survey: the name of the service X. In this case, X is "MalService". This is the only service could be executed in the future calling process. I guess the program might also contain function CreateService () and the functions related to mutex to make sure that only one process is running. We will check it later.

After the calling of StartServiceCtrlDispatcher, the program called sub_401040 at address 0x401032, I guess this is the function where the service is implemented. We double click on it. There are a lot of APIs here. The first API call at the this code area is OpenMutex() at 0x401052. The function is to open an mutex, which is different from CreateMutex(). The reason why the program uses the former one instead of the later one is that CreateMutex will create a mutex if the certain mutex does not exits. In this case, the certain mutex is HGL345. And it does not exist so far. The purpose of program so far is to check the existence of mutex rather than create a mutex. So, the program uses OpenMutex. Among the three parameters of this function, the most important on is Inherit Handle and IpName. IpName is the certain handle the program intends to open. The Inherit Handle value is a flag. If the value of Inherit Handle is true, then the process will inherit the handle and return the handle value. In this case the flag is 0, so the process will not inherit the handle. The return value will be NULL, which means the function is failed and the current Mutex HGL345 does not exist. Here we want the function fail and return 0 because the program has not created the mutex. If the mutex existed, the program then would exit, see function call ExitProcess at 0x40105E. Since the function return 0, the program will jump to location 0x401064. I am sure that the code area of the address contain CreateMutex(). Double click on loc_401064.

At 0x40106E, 0x40107A, we noticed CreateMutex() and OpenSCManager(), respectively. The mutex HGL345 is created after the call of CreateMutex. Then, the program establishes a connection to this malware control manager on the specified computer and opens the specified database.

At 0x401082, GetCurrentProcess returns the pseudo handle for the current process. GetModuleFileName returns the full path where the program has been executed. The return path is denoted as X which is used for the next function call. At 0x4010BC, the function call CreateService() helps create a service at the corresponding location with X and add it to service control database. Most of the parameters in CreateService have value 0. But three of parameters are different. dwStartType has value 0x02, which indicates the service starts automatically by the service control manager during system startup. dwServiceType is 0x10, which indicates that the service runs in its own process. The dwDesiredAccess is 0x02, which indicates the access token value. Now we can answer this question. The program set the StartType of the process to 0x20. Therefore, the process will run automatically each time the system restarts.

2. Why does this program use a mutex?

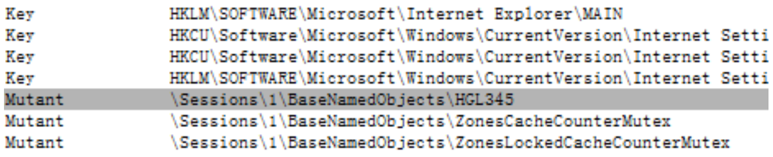
There are two functions related to mutex.OpenMutex() and CreateMutex(). OpenMutex is to check if HGL345 exists while CreateMutex is used for creating HGL345.The first one is indispensable because if HGL345 has already run, then the function would return success and the program would stop. Therefore, the main reason to use a mutex is to make sure that only one executable process is running.

3. What is a good host-based signature to use for detecting this program?

We run the malware on virtual machine and we can launch process monitor or Windows Explorer to search for the host-based signature. We notice the mutex name HGL345 and a service named Malservice. Both of them indicate the malware we are running.

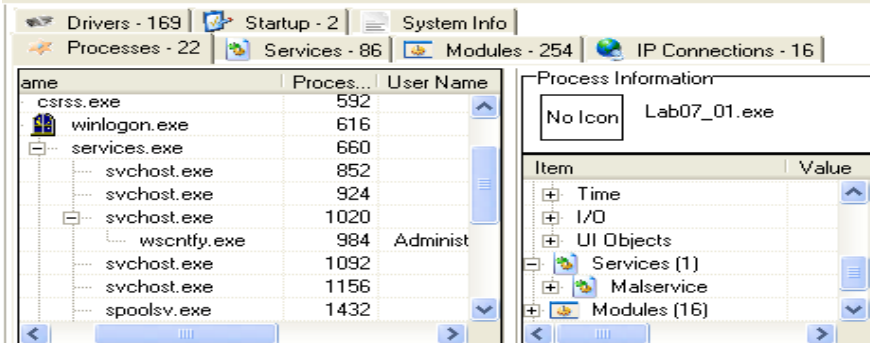
```
see location 0x4010B1      push    offset DisplayName ; "Malservice"
             0x401065      push    offset Name     ; "HGL345"
             0x40106A      push    0               ; bInitialOwner
             0x40106C      push    0               ; lpMutexAttributes
             0x40106E      call    ds:CreateMutexA
```

Therefore, HGL345 and MalService could be used to detecting this program.



```
Key         HKLM\SOFTWARE\Microsoft\Internet Explorer\MAIN
Key         HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Setti
Key         HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Setti
Key         HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Setti
Mutant      \Sessions\1\BaseNamedObjects\HGL345
Mutant      \Sessions\1\BaseNamedObjects\ZonesCacheCounterMutex
Mutant      \Sessions\1\BaseNamedObjects\ZonesLockedCacheCounterMutex
```

But I also meet a problem when I ran the malware. It should be a service named Malservice. However, I tried to use Process Explorer and Whats Running to detect the service and I didn't find it. So I wonder if the malware has not been installed at the machine. And I wait for like few minutes meanwhile the service showed up.



ame	Proces...	User Name
csrss.exe	592	
winlogon.exe	616	
services.exe	660	
svchost.exe	852	
svchost.exe	924	
svchost.exe	1020	
wscntfy.exe	984	Administ
svchost.exe	1092	
svchost.exe	1156	
spoolsv.exe	1432	

Process Information

No Icon Lab07_01.exe

Item	Value
Time	
I/O	
UI Objects	
Services (1)	
Malservice	
Modules (16)	

4. What is a good network-based signature for detecting this malware?

We haven't seen anything related to network so far. Hence, we should keep going through the function calls after CreateService. At 0x4010E5, there is a function call SystemTimeToFileTime() that convert the system time to file time. It also has a lot of parameters such as systemtime. second, systemtime.dayofweek,systemtime.hour, and systemtime.year. This code area sets the neither the starting time or the stopping time. Since the program has already started, we can make sure that it is stopping time. The malware will not stop until the time is up to the certain date.

If we simplify the code like the following:

```
xor     edx, edx
mov     DayOfWeek, edx
mov     Hour, edx
mov     Second, edx
mov     Year, 0x834; 0x834=2100 in decimal
```

The value in edx is 0. Only the "year" has another value. So, the specific time for the malware to stop running should be 0,0, 0, 2100, which is equivalent to the first day first seconds at 2100. January 1st, 2100. 00:00am. The following function calls like CreateWaitableTimer() and SetWaitableTimer() also indicates that the malware will wait for a long time before stop.

Waitforsingleobject() at 0x401110 checks the current state of the specific object. If the object's state is not signaled, the calling thread enters the wait state until time is out. In this case, the wait time is 0xFFFFFF seconds. At the location 0x401118, we saw a jump condition that if the return value is not zero, then the program will jump to location 0x40113B. Actually for this function, we don't expect the zero return value. We hope the value is 0 because the function returns 0 only when the mutex is abandoned or the function call has failed. The return value basically indicated the reason why the program success. If the function returns value is not 0, the program will jump to location 0x401126 where the program stays in sleep status for 0xFFFFFFF milliseconds.
Let's talk about the condition when the waitforsingleobjects() failed, then the program calls CreateThread() and move 0x14(20 in decimal to esi). We keep going through the program and we would notice that there is another CreateThread() at the location of 0x401135.At the two lines after the second CreateThread() function call, the program jump to location 0x401126 again while the second

CreateThread() function call is in the location of 0x401126. Now, we know that it is a for loop.like the following:

```
CreateThread()
for(i=14;i<0;i--)//i-- is corresponding with "dec esi"
```

The register esi here is equivilant to "i". The function CreateThread() has a lot of parameters but only one parameter has value.

lpStartAddress here is not a string pointer. It is a pointer to the application-defined function to be executed by the thread. This pointer represents the starting address of the thread. In another word, it is a function pointer. Besides that parameter, the lpParameter is also important because it is a pointer to a variable to be passed to the thread.
To see the function pointed by StartAddress, we double click on it. Finally we saw the network-based indicator after we click it. The codes are somehow similar to the previous lab6.

Internet Explorer 8 at 0x40115A is the user agent which is used for future download/upload at InternetOpen().

At the code area of 0x40116D, we notice another network-based indicator. http://www.malwarepracticalanalysise.com

5. What is the purpose of this program?

The function call InternetOpenUrl is right after the second network-based indicator. The URL http://www.malwarepracticalanalysise.com is the parameter to the function call. After the function call that opens the URL, the program starts an infinite loop within the code area of 0x40117D.

There is no way to break this infinite loop until the starting date of 2100 when the program will not create mutex, service and thread. The program will die automatically. But before the first date of 2100, the program cannot stop running. It sends request to open the certain URL. In this case, http://www.malwarepracticalanalysis.com. I assume the program is trying to paralyze the website by continuously sending request. So, the ultimate goal of the program is not to open this RUL but destroy it and forbid other uses to browse it.

6. When will this program finish executing?
At the beginning date of 2100.
00:00:00 am, 01/01/2100.

Lab7-02
1. How does this program achieve persistence?

If we analyze this program in the same way we analyze the previous lab, we can understand the function of this malware.

We go through this program by checking the API functions. The first function call is OleInitialize() at 0x401005 to initializes the component objects model library on the current unit. If the return value is negative, the initialization process fails and then the program will terminate. Otherwise the program will prepare for another important function call CocreateInstance() which is used to get access to COM functionality. CocreateInstance() takes five parameters:

```
push    eax;     pointer variable that receives the interface pointer requested in riid.
push    offset riid;      riid is interface identifier
push    4;       dwclscontext
push    0;       pUnkOuter
offset  rclsid;  rclsid
```

pointer virable(ppv) and ppv are more important than other parameters because ppv stores the request sent by riid. It could tell us what dose the malware do. I guess that riid contain a command or a list of commands. The ppv will retrieve the command and then execute it. To see the value at riid and rclsid, we double click on them while the value seems like unreadable.

riid has value D30C1661CDAF11D08A3E00C04FC9E26E
rclsid has value 0002DF0100000000C000000000000046

If we look up the riid value at Google, we will learn that the value indicates the interface type and relates to registry. So, we need to use registry editor to see more detail of the riid value and rclsid value.

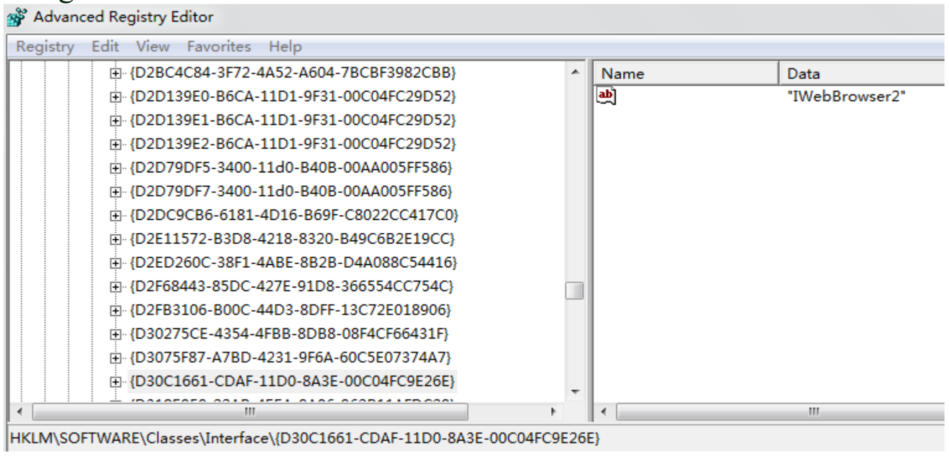
So far I don't think this program can achieve persistence because I didn't notice any for loop or while loop. The only function could be executed more than once is

OleInitialize(). After the function CoCreateInstance(), if the function has failed and returned 0, the program will call OleInitialize() again and then exit. The function call located at 0x401050 SysAllocString is to allocates a new string and copies the passed string into it. Then the program moves pointer and address within different registers and call SysFreeString which is a pair with SysAllocString. None of them involve loop and persistence token. Therefore, there isn't anyway to achieve the persistence here.

2. What is the purpose of this program?

First, run the executable file.

At the previous question, we said registry editor could give us more information. The interface key riid is located at HKLM\SOFTWARE\Classes\Interface. So we should navigate to this area and find D30C1661CDAF11D08A3E00C04FC9E26E



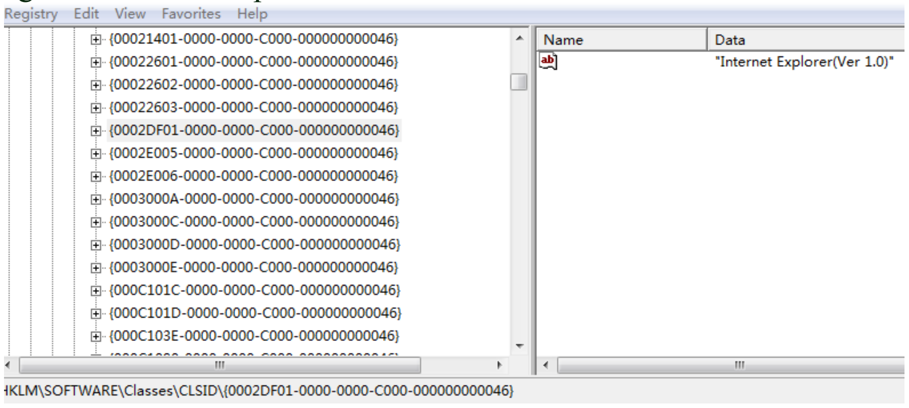
```
{D2BC4C84-3F72-4A52-A604-7BCBF3982CBB}
{D2D139E0-B6CA-11D1-9F31-00C04FC29D52}
{D2D139E1-B6CA-11D1-9F31-00C04FC29D52}
{D2D139E2-B6CA-11D1-9F31-00C04FC29D52}
{D2D79DF5-3400-11d0-B40B-00AA005FF586}
{D2D79DF7-3400-11d0-B40B-00AA005FF586}
{D2DC9CB6-6181-4D16-B69F-C8022CC417C0}
{D2E11572-B3D8-4218-8320-B49C6B2E19CC}
{D2ED260C-38F1-4ABE-8B2B-D4A088C54416}
{D2F68443-85DC-427E-91D8-366554CC754C}
{D2FB3106-B00C-44D3-8DFF-13C72E018906}
{D30275CE-4354-4FBB-8DB8-08F4CF66431F}
{D3075F87-A7BD-4231-9F6A-60C5E07374A7}
{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}

"IWebBrowser2"

HKLM\SOFTWARE\Classes\Interface\{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}
```

It tells us the interface is IWebBrowser2 that exposes methods that are implemented by the WebBrowser control. The pointer variable (ppv) stores the interface pointer requested. Therefore, we can assume that WebBrowser2 contains a few commands. Later we will determine which command is being used by the malware.

The class identifier rclsid is located at HKLM\SOFTWARE\Classes\CLSID. We navigate to this area and find 0002DF0100000000C000000000000046. It tells us the user agent is Internet Explorer Version 1.0.



```
{00021401-0000-0000-C000-000000000046}
{00022601-0000-0000-C000-000000000046}
{00022602-0000-0000-C000-000000000046}
{00022603-0000-0000-C000-000000000046}
{0002DF01-0000-0000-C000-000000000046}
{0002E005-0000-0000-C000-000000000046}
{0002E006-0000-0000-C000-000000000046}
{0003000A-0000-0000-C000-000000000046}
{0003000C-0000-0000-C000-000000000046}
{0003000D-0000-0000-C000-000000000046}
{0003000E-0000-0000-C000-000000000046}
{000C101C-0000-0000-C000-000000000046}
{000C101D-0000-0000-C000-000000000046}
{000C103E-0000-0000-C000-000000000046}

"Internet Explorer(Ver 1.0)"

HKLM\SOFTWARE\Classes\CLSID\{0002DF01-0000-0000-C000-000000000046}
```

Compared to this entry, the entry of interface key is more useful. The reason why we have to find out the value stored at ppv is that the value transfered to edx and implement by the malware at location 0x401074.

If we check codes in reverse order:

```
	0x401074	call dword ptr [edx+0x2C];0x2C=44;
	0x401065	mov edx, [eax]
	0x40105C	mov eax, [esp+0x28+ppv];
```
The value at ppv is passed to eax and then edx. The program implements the function pointed by ptr[edx+0x2C]. We check the interface IWebBroser2 so that we can learn it has a lot of methods. The corresponding functions to those methods are stored at either exdisp.h or objbase.h. Both of them are common related to interface function. We search both of them by applying key words "WebBrowser2". Eventually we found it at exdisp.h.

One pointer takes 4 bytes. Therefore we do the math to determine which method is used by the program.

0x2C/4+1=12(in decimal); add 1 because the pointer address start at 0.

The 12th entry at the IWebBrowser structure is like the following:

STDMETHOD(Navigate)(THIS_BSTR,VARIANT*,VARIANT*,VARIANT*,VARIANT*)PURE;

We check the methods table for IWebBroser2, Navigate tries to access to a resource identified by a URL or to a file identified by a full path. Here the URL is passed by SysAllocString. The value is http://www.malwareanalysisbook.com/ad.html.

Later the value will be stored at ecx. When the program calls ptr[edx+0x2C], the navigate function then open the URL webpage. Since we have already run the executable file, we know that it is an advertisement page. Therefore, the goal of this malware is to launch an advertisement page. The function CoCreateInstance() is important to the malware, which helps the malware open the advertisement webpage.

3. When will this program finish executing?

I think this malware just runs once. After the advertisement page is launched, the malware will exit itself. No time stamp. No time frame indicator.

Lab7-03

1. How does this program achieve persistence to ensure that it continues running when the computer is restarted?

We still pay attention to function calls in this program. At the beginning part of the main function, the codes indicate the correct way to run the malware. It compares number 2 and the value stored in argc which means the number of input strings. Then, it compares the first and second value stored in argv(pointer to input string) with the certain string word by word. If any of the comparing result is not equal to 0, the program will exit. Otherwise it can keep running. At the code area of 0x401460, the program compares our input string with "program name WARNING_THIS_WILL_DESTROY_YOUR_MACHINE". The first parameter is stored at register dl, the second parameter is stored at register bl. In this case, program name is Lab07-03.exe. Therefore, the right way to launch the malware is to open command line and then type Lab07-03.exe WARNING_THIS_WILL_DESTROY_YOUR_MACHINE

The first function call CreateFile() is located at 0x4014AC rather than 0x401495 where the parameter to the function is passed. The malware creates a file named kKERNE123.dll at path C:\Windows\System32 on purpose because KERNE123.dll is really similar to KERNEL23.dll. When the file is created, CreateFileMapping() and MapViewOfFile() help allocated virtual memory to the dll file. CreateFile() is called again at 0x4014F0. But this time the program opens Lab07-03.dll and also allocate memory to the file by applying CreateFileMapping() and MapViewOfFile(). The next function call appears at 0x40164D where the program jump to the CloseHandle() if ebp is equal or less than 0. Between the code areas of 0x40164D and 0x4014F0, the operations are so complicated that I have tried to understand it but I just couldn't manage to do it. The only thing I can figure out is that the program tries to allocated memory and write memory for future use. At the codes area of 0x40164D, the function CloseHandle() has been called twice because it closed the handle of lab07-03.dll and KERNE123.dll as well. Then the malware performs a crucial step that copy file Lab07-03.dll to KERNE123.dll. Therefore, we can assume that KERNE123.dll is infected by Lab07-03.dll because it is equal to Lab07-03.dll functionally. If the function call failed, the program exits other wise the program will jump to 0x401806 where the program calls sub_4011E0 with parameter "C:\*". When I saw the asterisk, I guess the function performs a searching method. We double click on it and check what's going on.

Here the first function call we noticed is FindFirstFile(). The parameter ".exe" is divided into two parts "." and "exe" when it comes to further analysis. The format of the file is like "fileName"+"."+"fileAttribute". The fileAttribute is exe. For example, the FindFirstFile() find a file named "random.exe", the program will check if the path of random.exe is located at C:\ and if the format of the file is "random"+"."+"exe". After the comparison and search, the program called Stricmp at location 0x4013F6.

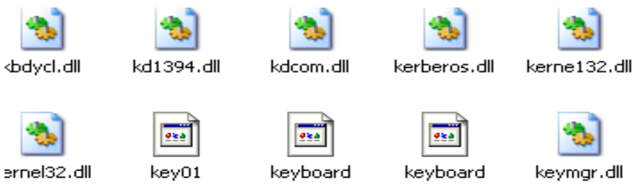
Stricmp performs a lowercase comparison between strings. The two parameters here are passed by offeset ".exe" and register ebx that stored the file attribute. It checks if the file at C:\ is executable file. If the file is an executable file, the program will go back to main function and perform the malicious task that import the dll file. Otherwise it calls FindNextFile() to find the next executable file. If it finds an executable file, then goes back to 0x401210 where the FindFirstFile() is just called and the searching task is still carrying on. The function here is pretty similar to recursive that the program searches every executable file at C:\ before it ends.

During the above process, the program would exit even though there is only one function call failed. So, the malware could achieve persistence by infecting all the executable files at C:\ and import the dll file to those infected executable file.

However, when I ran the malware on machine, the malware exit quickly. Therefore, I doubt if the malware can really achieve persistence. Besides, I still have a lot of questions that I cannot understand in the Lab7-03.exe file for exmaple how does the program map memory and address. But I will take time to figure it out.

2. What are two good host-based signatures for this malware?

The first host-based signature is KERNA123.dll at C:\Windows\System32. We can run the malware and check the lower panel showing the dll and mutex like we did at previous lab. As we talked above, the malware can only be executed by the correct input at command line. The input should be Lab07-03.exe WARNING_THIS_WILL_DESTROY_YOUR_MACHINE. I run this command and I can see lab07-03 in process monitor. However, it quit after few seconds. The running time is so fast that I cannot make a screenshot and check the lower panel. But the executable file was right there. So, I open C:\Windows\System32. I saw the dll file which is similar with kernel32.dll.



To see the second one, we should open Lab07-03.dll by IDA pro. We still follow the order of function calls. The first function call is CreateMutex(). The program creates a mutex with Handle SADFHUHF. The parameter is passed at 0x10001048 and 0x10001067 within function call OpenMutex() and CreateMutex, respectively. The reason to create Mutex is to make sure only one copy of the program is running. The second function call is WSAStartup() that initiates use of the Winsock DLL by a process. I guess the program is going to connect to Internet and then create a process. After few lines, we noticed Socket() and Connect() which definitely open a socket and starts conversation. The Internet address 127.26.152.13 is converted to numeric address by inet_addr(). And it is converted from the unsigned short integer hostshort from host byte order to network byte order by htons(). Now the malware operates send() and recv() functions to communicate with the victim machine. Send() has a parameter labeled as "hello" which is sound like a starting greetings to the victim machine. Recv() has a parameter stored at eax. Here the parameter is not the string "Hello". See that the parameter is stored at [esp+0x120+buf] where the stack stores the network traffic command. The address [esp+0x120+buf] only appears at that location. At the first place I cannot assure the value inside of it. But if we keep moving to next few lines, we noticed the function call strcmp() and parameter "sleep", which means "sleep" is compared to a string "X". Here X is stored at the pointer buf. So, I am sure that a network traffic command "sleep" is passed by recv(). If the traffic command is sleep, the program will sleep for 0x60000 millisecond which is 6.5 minutes in decimal. If the traffic command is not sleep, the program jumps to 0x10001161.

At 0x10001161, the network traffic command is compared to "exec". If it doesn't match, then it will compare with "sleep" again. I guess the traffic command is either exec or sleep. If the two strings match, the malware create another process by CreateProcess() at 0x100011AF. To find out the performance of the process, we should check the parameters. CreateProcess() has a lot of parameters. The most important two are lpApplication and lpCommandLine. lpCommandLine is the command line to be executed. If lpCommandLine parameter is NULL, the function uses the string pointed to by lpApplicationName as the command line. Unfortunately, I cannot determine the value stored in lpCommandLine. However, I guess the content here is a path of the executable file because lpApplication is used for searching the file with a certain extension. It specifies the file module, which infers that lpApplication and lpCommandLine somehow perform similar searching task. So the value in lpCommandLine is path while lpApplication is the attribute of the file, for example ".exe" or ".com".

The second host-based indicator is mutex SADFHUHF. But like I said above, I am not able to see the malware in process monitor. I cannot see the mutex.

3. What is the purpose of this program?

Infected the executable file in C:\. Normally dll file performs as backdoor. In this case, Lab07-03.dll and KERNE123.dll are the back doors that root in the executable files at local machine. The backdoor will connect to remote network 127.26.152.13 and send hello to victim machine. After the communication, the malware either preform sleep or execute the command line.

4. How could you remove this malware once it is installed?
I don't think the malware could be removed because all of the executable files have been infected. It is impossible to remove all the exe files at machine.