Xiaoyi Zhou

Purdue University CIT581

Malware Forensics

Lab 9: OllyDbg

Due October 23, 2014

Instructor: Samuel Liles

**Abstract**

This lab is a re-written lab because the first version is awful. It is my second time to analyze lab9. The procedure becomes easier than the first time I launch OllyDbg. The first lab is a backdoor that requires password to launch. Once it is launched, it starts a reverse backdoor and executes the command via network. The second lab starts a reverse shell that starts communication between socket and cmd.exe. The third lab pings to the specific URL every day at 1:00AM. I think the third lab is the most interesting one. I have to say that the analyzing process is easier a lot if we do it again.

**Lab09-01**

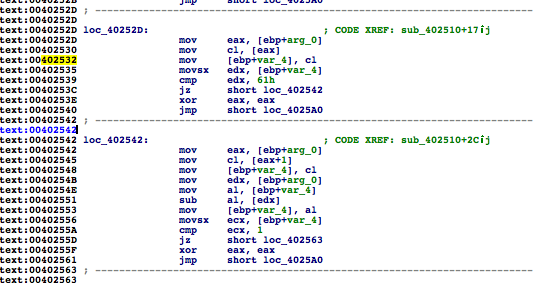
**Steps of Processes**

We load the malware into IDA Pro for basic analysis. At the code area of main function, the first function call is sub\_401000 where the program intends to open the registry key SOFTWARE\Microsoft\XPS. If we check the registry, we will know that this entry doesn't exist. So the function call failed at that time. The program will jump to next function call sub\_402410 that the malware deletes itself. The program will start a command shell and then run /c del> nul to delete the program. To avoid that the program kills itself, we should detour it. The registry value doesn't exist right now. So we cannot let the program execute sub\_401000 anyway. At two lines before sub\_401000, the program checks the input with value one. If the input is nothing, then the program will run sub\_401000. Therefore we should type something instead of nothing. Assuming we have already passed the delete command, the program jumps to loc\_402B1D.



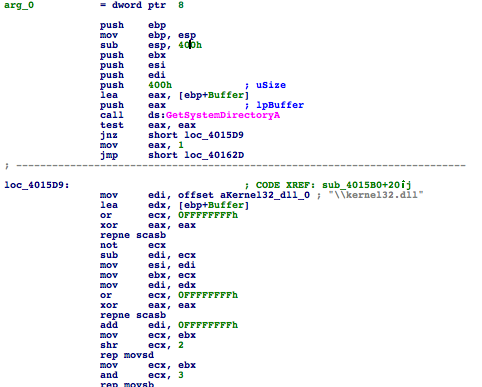
At loc\_402B1D, the program calls sub\_402510 that check if the input string is correct. argv is the start pointer to the input string. argc is the numbers of characters. The function takes [argv+argc\*4-4] as parameter. Therefore the input string has length four. The parameter is the last character of the string. Now the last string is passed to sub\_402510, the function will do reverse checking. Check the start of sub\_402510, the function has two parameters, var\_4 and arg\_0. var\_4 is the last character. arg\_0 is the start pointer to the string. There are not equal right now. But they become equal at 0x402532 because the program moves the value from arg\_0 to var\_4. We can analyze the comparison of the first character and second character because the rest characters use same methods. The first character is compared with 0x61 which is 'a' in ASCII. If it is 'a', then the program jumps to next comparison. The eax is added by one, which means the pointer is moved to right by one. The pointer is pointing to the second character. The second character stored at al subtracts the first character stored at edx. If the result is equal to one, which means the second character is letter 'b', the program jumps to next comparison. At loc\_402563, the third letter is compared with 'c'. The last letter subtracts 'c' and then compare with number one.

The purpose of sub\_402510 is pretty clear. If the input string is 'abcd', the program can keep running otherwise it will stop.

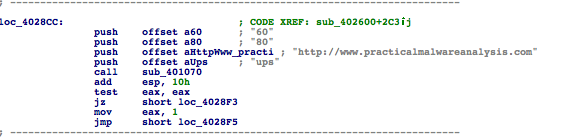


The program jumps to loc\_402B3F if the input is correct. At loc\_402B3F, the program calls mbscmp to check the input string again. This time the program checks the input with command "-in" which is the standard command to install a malware. If the command operation is not correct, the malware will still delete itself otherwise it will start a service. The service name is a parameter passed to sub\_402600. We can double click on sub\_402600. The program tries to access to the open a service because the program calls OpenSCManager. The parameters of the function are a little bit strange because there isn't any specific string indicating the service name. IDA Pro helps us label the service name as lpDataBaseName, which means the service name is the base name of the malware. At 0x402632, the program builds the string %SYSTEMROOT%\system32\. At 0x40268F, the program concatenates the string with ".exe". The name of the executable file is the name of the program. The program creates service with binary path C:\WINDOWS\System32\filename.exe.

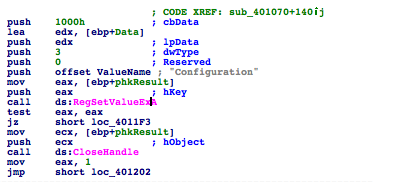
We can keep going through the codes until we find the next important function CopyFile. The function takes destination and source parameter. Here the source file is the malware. Destination file is the system path same as the service path. The malware copies itself to C:\WINDOWS\System32.



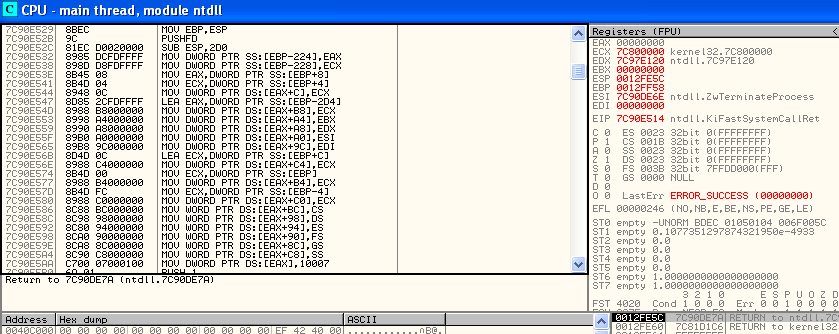
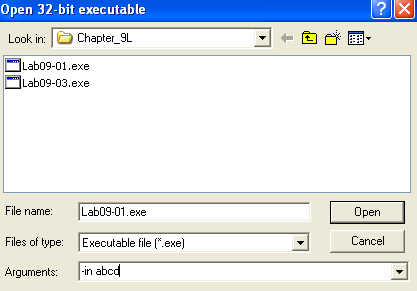
After the malware copies itself, the program retrieved the system directory and concatenated the path to kernel32.dll. The purpose is to get the path to kernel32.dll. Then the paths of kernel32.dll and the copied malware are passed to a new function call sub\_4014E0. We double click on the function. There are some modifications on time stamps to the copied malware. The program calls GetFileTime and SetFileTime to change the time stamps of copied file. After the modification is applied, the copied file will have the same time stamps with kernel32.dll.

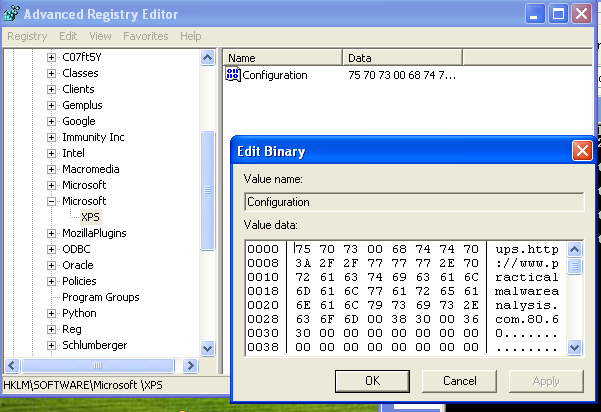


At sub\_401070, we see the first network indicator. The URL and ports are parameters passed to sub\_401070. We double click on it to see more details. The information related to the URL and the ports number is regarded as registry data. The program create register key SOFTWARE\Microsoft\XPS by calling RegCreateKeyEx. Now the registry key exists, the program changes the value by calling RegSetValueEx.The info related to network is stored as configuration value. We will verify the value after we put the malware into OllyDbg later.



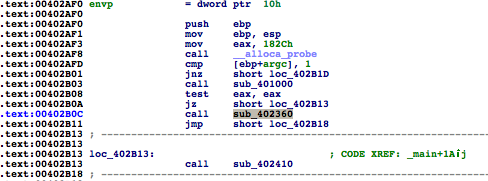
We put the malware into OllyDbg and set the input as -in abcd. The malware runs successfully.





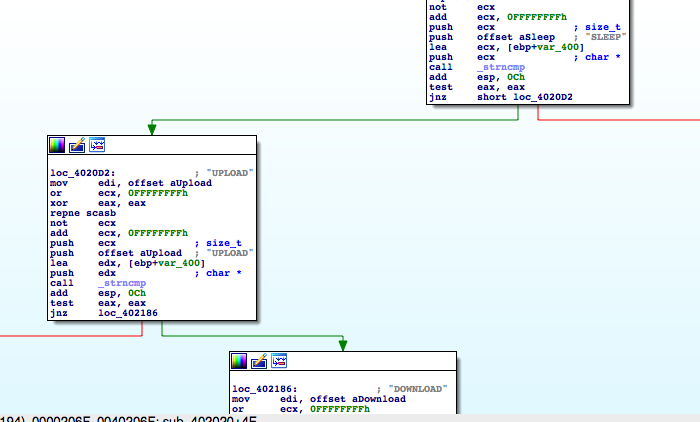
See the configuration data has been changed to the network related information. The analysis is not over yet. Please the issues or problems section.

**Issue or Problems**



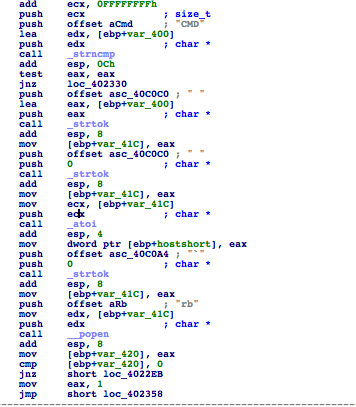
Most of network-related functions are located at sub\_402360. However I don't think this function will be executed if the input string is correct. There is a comparison between input string and integer one. The program will execute sub\_402360 only the comparison is successful. However the correct installation string is "lab09-01.exe -in abcd". The comparison will never succeed. We can double click on sub\_402360 see the details of the network-related functions. From the graphic view we can guess that the program will retrieve a series of number that is associated with network command. Then the program compares the string with the command. It will execute different function according to the corresponding command. The first command we saw is SLEEP. The program calls strtok to eliminate the space. The result will be a string for example "30". It is not an integer number but a string. The string will be converted into number by calling atois. In this case, the converted number is 30 seconds. Hence the program will sleep for 30 seconds

The next command is UPLOAD. This function passed the port number and the URL to WSAStartup and gethostbyname in order to start a socket. Once the socket start, the program immediately create file at local path. The file receives the data communicated via the socket by calling recv.



The next command is DOWNLOAD. This function takes the lpFileName, port number and the URL as parameter. It also starts a socket at 0x4017AC and gets access to the file at local path. At loc\_4018E3, however the function is really different with UPLOAD because it reads the file instead of writing data to it. If read function is successful, the program will jump to 0x401945 where the program calls send function in order to send data to the remote server. So DOWNLOAD is opposite to UPLOAD.

The next command is CMD.



We should be careful with this function. The program takes "rb" and "CMD" as parameter and then calls popen. It didn't open a file but a pipe stream to/from a process. The process is cmd.exe. Therefore the program actually starts the cmd.exe in order to transfer data. After that, the program builds the server again at 0x4017AC. This function has been called many times. We can rename it as Start\_Server. When the server is built, the program calls fread and send the data to the server.

The last command is NOTHING, which indicates that the program did nothing when the command is executed.

This lab is a little bit difficult. However if we put it into IDA Pro first and then analyze it by OllyDbg, we can solve a lot of problems. The key to solve analysis the malware is to figure out the password and command options. But I am confused about the network function that I mentioned before. sub\_402360 will not be execute unless the input is not same as "Lab09-01.exe -in abcd". However the program will delete itself if we don't input the correct string. We can bypass the password by patching the binary. However the password is still necessary for the future analysis.

**Conclusion**

This malware can only be launched if we provide correct password "abcd". The malware copied itself to C:WINDOW\System32. It got access to register key and create a service. We can remove the malware by using "-re" command option. To execute the command option, password is required. Therefore patching the binary to bypass the password is not a good option.

**Reviewed Questions**

**1. How can you get this malware to install itself?**

We can use '-in' plus the password 'abcd'.

**2. What are the command-line options for this program? What is the password requirement?**

The password is abcd. There are four command line options. '-in' indicates the malware to install itself; '-re' means delete itself; '-cc' means print the configuration;'-c' means update the configuration.

**3. How can you use OllyDbg to permanently patch this malware, so that it doesn’t require the special command-line password?**

We can batch the binary to set the return value to TRUE. Change the value at eax to one and return it. So it will always be true.

**4. What are the host-based indicators of this malware?**

The malware got access to the register key at HKLM\Software\Microsoft \XPS\Configuration. Moreover, it also copied itself to the C:\WINDOWS\System32.

**5. What are the different actions this malware can be instructed to take via the network?**

There are five options for the malware. Sleep, Upload, Download, CMD, Nothing.

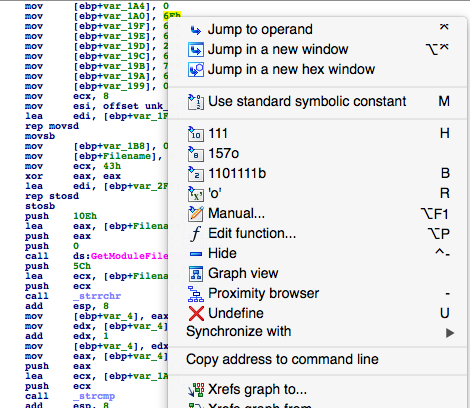
**6. Are there any useful network-based signatures for this malware?**

We can see the network beacon http://www.practicalmalwareanalysis.com/ and HTTP/1.0 GET request.

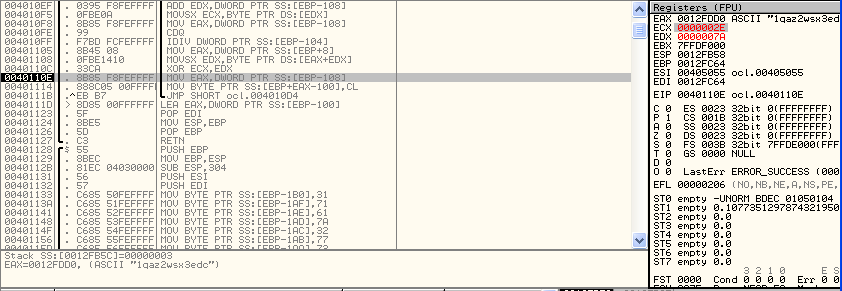
**Lab09-02**

**Steps of Process**

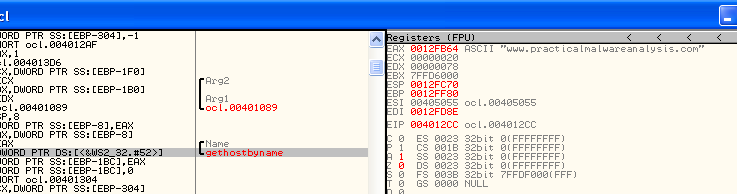
We load the malware into IDA Pro and OllyDbg. The first thing I noticed is there are a lot of MOV operations. The MOV operations are used to format the string(s). In this case, the program formats two strings. Because the string should be ended with terminate character NULL which is '0'. We can get the string by right click on the characters. The first string is 1qaz2wsx3edc. The second string is ocl.exe. So far the first string is not readable. I think it might be the password or the ciphertext. The first function at the program is GetModuleFileName. The function will return the name of the malware. Then the malware takes the filename and 0x5C ('\' in ASCII) as parameter to call strrchr. The function is used to find the specific string '\Lab09\_02.exe'. However after the string pointer pointing to the string, the program compares the string with character "o". EDX is increased by one because the program wants to eliminate "\" character. If the comparison failed, then the program will exit immediately. If the comparison success, the program will jump to loc\_40124C. Therefore in order to launch the malware correctly, we should rename the malware with name starting with "o". The strange thing is I rename the malware as "oo.exe", it still can run. There isn't any loop. So it is supposed to run. I have to change the name of the malware as ocl.exe in order to launch it.



The malware calls WSASocket and Gethostbyname to intercept with network. Before the malware called Gethostbyname, it calls sub\_401089. The function is supposed to return an address that can be used to Gethostbyname. The function sub\_401089 takes a pointer pointing at [var\_1B0]. It is the start of the first string 1qaz2wsx3edc. Now I get it why I have to change the program name to ocl.exe even though there isn't any comparison loop because EDX is also a string pointer. sub\_401089 should be a decryption function used to decrypt the first string. We double click on it. The cipher schema is XOR at 0x40110C. To decrypt the string, we can write script or use OllyDbg. Since this chapter is about OllyDbg, we should try OllyDbg. The program XORs ecx and edx. The value in edx is the ciphertext. However after XORs, the plaintext character is moved to ecx. So we should pay attention to ecx instead of edx. We set a breakpoint at 0x40110E to see the plaintext. At the first time the program hits the breakpoint, the value at ecx is 0x77, so is the second time and third time, which means the first three characters are www. The next time the program hits the breakpoint, the value at ecx becomes to 0x2E, which represents "www.XXXXXXX".



The length of the plaintext should be 32 according to the analysis from IDA Pro. After the ciphertext is decrypted, the plaintext is stored at eax at 0x40111D. The plaintext is passed to gethostbyname at 0x4012CC. To see the full plaintext, we can set a breakpoint at 0x4012CC. Now we can see the plaintext is www.practicalmalwareanalysis.com.



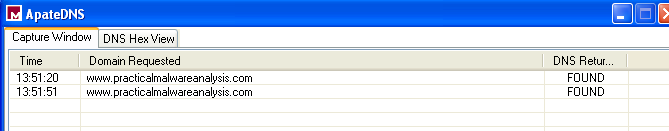
If gethostbyname failed, the program will sleep for 30 seconds and go back to the place where WSAStartup is called. The program will basically start over until gethostbyname successes. If the function call doesn't fail, the program will jump to loc\_401304.

At loc\_401304, the program calls htons and connect. The program is trying to connect with www.practicalmalwareanalysis.com. If the connection failed, the program will sleep for 30 seconds and go back to loc\_40124C where the WSAStartup is called otherwise the program will jump to loc\_40137A.

At loc\_40137A, there is a really important function call sub\_401000. We double click on it and start analyze. At first glance, we can guess that the malware will start a shell because we can see it calls CreateProcess with parameter cmd.exe. But there are a lot of parameters before that. So we should figure out those parameters. The parameters are the value of STARTUPINFO structure. The program set the value of StartupInfo.wShowWindow to zero, which means the program will hide the window and activates another window. StartupInfo.hStdInput is set to edx. edx stores the value at [arg\_10] which is the return value of the previous function call connect. The return value of connect should be a socket. Therefore StartupInfo.hStdInput is set to socket. Both StartupInfo.hStdError and StartupInfo.hStdOutput are set to socket. Therefore all the data communicated(input) via socket will be sent to cmd.exe; all the data communicated(output) via cmd.exe will be sent to socket.



We cannot see the window for cmd.exe. But we can clearly see that the malware got access to www.practicamalwareanalysis.com



**Issue or Problem**

I have a question for the decryption function. Even though I can decrypt it, I cannot figure out what is the value of the cipher key from IDA Pro. And I cannot find "Follow the dump" for edx.

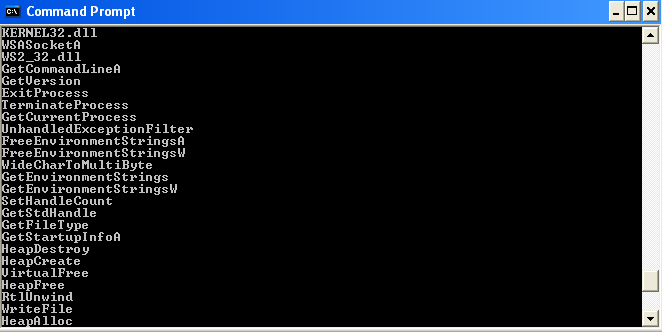
**Conclusion**

This malware starts a revere shell that sends data between socket and cmd.exe.

**Reviewed Questions**

**1. What strings do you see statically in the binary?**

There are a lot of imports function if we check the strings.

****

**2. What happens when you run this binary?**

Nothing shows up. But I know it starts a hidden cmd window and access to www.practicalmalwareanalysis.com.

**3. How can you get this sample to run its malicious payload?**

We should change the name to ocl.exe

**4. What is happening at 0x00401133?**

The malware start format and build two strings.

**5. What arguments are being passed to subroutine 0x00401089?**

The ciphertext 1qaz2wsx3edc.

**6. What domain name does this malware use?**

www.practicalmalwareanalysis.com.

**7. What encoding routine is being used to obfuscate the domain name?**

XORs

**8. What is the significance of the CreateProcessA call at 0x0040106E?**

The CreateProcess function call starts a cmd.exe window. But the window is hidden because the STARTUPINFO parameter is set to 0. Moreover a lot of parameters at STARTUPINFO is changed so that the malware can start communication between socket and cmd.exe.

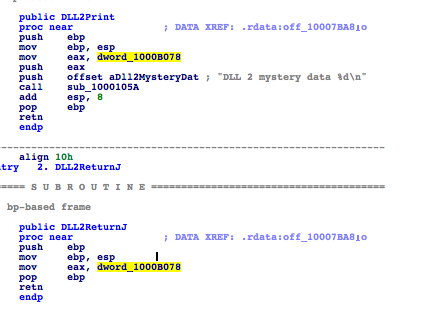
**Lab09-03**

**Steps of Processes**

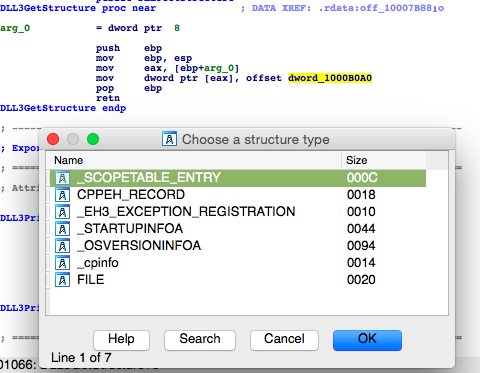
First we start statistic analysis. Check the imports table at PEView. The malware loads DLL1, DLL2, KERNEL32.DLL, and NETAPI32.DLL. However when I put the malware into IDA Pro, the first ting I noticed is DLL3. The malware also loads DLL3 by LoadLibrary, which means the DLL3 is loaded dynamically. We double click on the function. There are two references about the LoadLibrary. Besides DLL3, the malware loaded user32.dll dynamically as well. We need to check when and where DLL3 is loaded. Unlike DLL1 and DLL2, they are at import table so that they will be loaded as soon as the malware is launched. We also notice that the malware call DLL1Print, DLL2Print and DLL2Return. I assume DLL1Print and DLL2Print will give us a string related to a program. There is a WriteFile function call after few lines without CreateFile. Therefore we should consider if the previous function call will return a file name.



To verify our conclusion, we can load the DLL files into IDA Pro and check the exports function of them. The purpose of DLL1 is simple. The export entry print a string "DLL 1 mystery data %d\n". The number here is retrieved from DLLMain function. At DLLMain function, the program only did one thing. It calls GetCurrentProcessId to return the current PID. Therefore we can conclude that DLL1Print will give us the PID of the current running process. Now we load DLL2 into IDA Pro. The export entry point is DLL2print. The only thing DLL2 did is to create a file. If the file exists, then return the file handle number. I know it will not return the file name because it will print a string "DLL 2 mystery data %d\n". The integer here is retrieved from CreateFile. Therefore it won't return the file name. Remember that DLL2 has another export entry point called DLL2ReturnJ. The return value is stored at dword\_1000B078. If we click the variable, we will notice that the variable is used by the print function, which means the data stored at dword\_1000B078 is the integer indicating the created file "temp.txt". Hence the purpose of DLL2ReturnJ is same as DLL2Print.



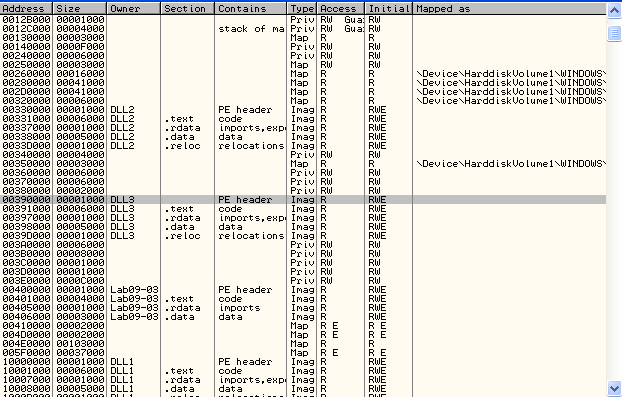
Now we go back to analyze the executable file. We already have the handle to temp.txt file. The written content is stored at office set. The content is the string malwareanalysisbook.com. We launch the malware and we can find the temp file at local path. Then the malware takes DLL3Print as parameter to Loadlibrary and GetProcAddress. It called GetProcAddress twice. The second time when the malware calls GetProcAddress, it takes DLL3GetStructure as parameter. So we should load DLL3 into IDA Pro. At the entry point of DLL3Print, the program print DLL 3 mystery data %d, the number will be the address of pointer WideCharStr. The pointer stored the string ping www.malwareanalysisbook.com. The string is converted at UTF-16 by MultiByteToWideChar.Next we check the entry point of DLL3GetStructure. It should display like a structure. But it is not. We can convert the variable dword\_1000B0A0 in to structure type by applying new structure type.

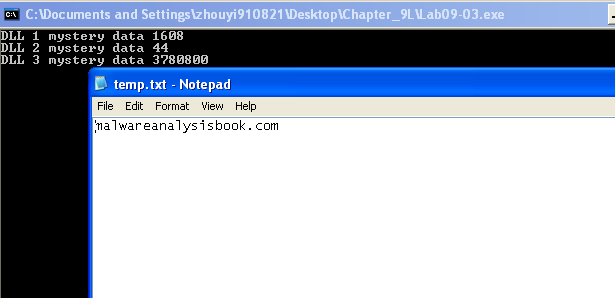


So far we don't which type we should apply. We can go back to the executable file. The next function call is NetScheduleJobAdd that submit a job to run at a specified future time and date. This function requires that the schedule service be started.The function takes buffer as parameter. The structure type of the buffer is AT\_INFO. Buffer first appeared at 0x401071 where the program takes buffer as parameter and then call [ebp+var\_10]. var\_10 stored the return value of GetProcAddress (DLL3GetStructure). Therefore the buffer is pointing to the structure of DLL3. We should add a structure type at DLL3 as AT\_INFO. Once we add it and change the variable dword\_1000B0A0 into the new structure type. We can notice that the program sets a time to ping the URL address. In other words, the malware will ping to malwarepracticalanalysis.com every day at 1:00 AM.



To see more details about the malware, we load it into OllyDbg. We navigate to the function call for Loadlibrary at 0x401032. We set a breakpoint after Loadlibrary to see the base address for the three DLL files. We notice all of them have base address 0x10000000, which matches with our statistic analysis.





The above screenshot is the result after the malware is launch. All of the data is corresponding with our analysis.

**Issue or Problem**

Once we can figure out the AT\_INFO structure for DLL3, the analysis procedure will be easy. For this lab, I think IDA Pro is more important than OllyDbg.

**Conclusion**

The malware dynamically load DLL3 file to ping practicalmalwareanalysis.com every day at 1:00 AM. It creates a temp file at local path to store the URL address. The function calls to execute the program and create file are stored at DLL1 and DLL2, respectively. Those two DLL files are loaded as long as the program is run because they are at imports table.

**Reviewed Questions**

**1. What DLLs are imported by Lab09-03.exe ?**

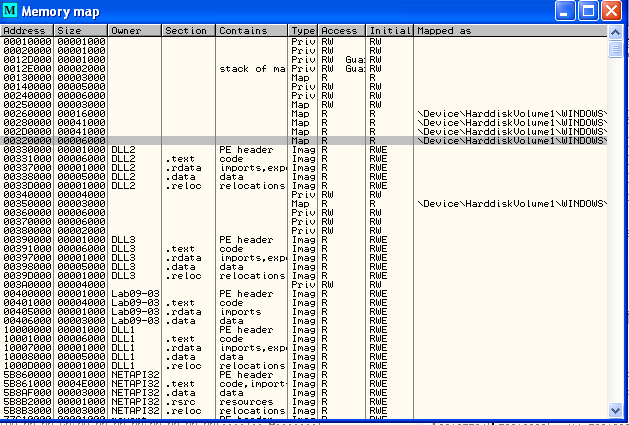
KERNEL32.DLL, USER32.DLL, NetAPI32.dll, DLL1.dll, DLL2.dll, DLL3.dll.

**2. What is the base address requested by DLL1.dll , DLL2.dll , and DLL3.dll ?**

The base address is same for them: 0x10000000.

**3. When you use OllyDbg to debug Lab09-03.exe , what is the assigned based address for: DLL1.dll , DLL2.dll , and DLL3.dll ?**

The assigned address is different with the base address. The assigned address for DLL1.dll is 0x1000000, for DLL2.dll is 0x320000, for DLL3.dll is 0x380000. To see the assigned address, we still need to set a breakpoint right after the Loadlibrary is called.



**4. When Lab09-03.exe calls an import function from DLL1.dll , what does this import function do?**

Print the string and return the PID.

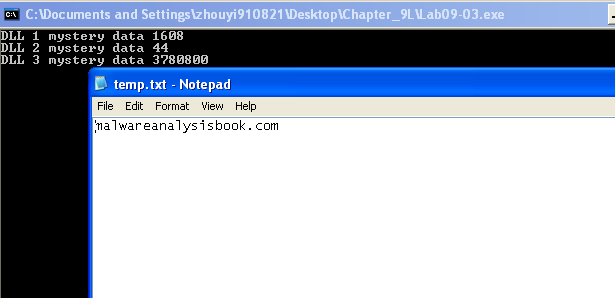
**5. When Lab09-03.exe calls WriteFile , what is the filename it writes to?**

temp.txt at local path

**6. When Lab09-03.exe creates a job using NetScheduleJobAdd , where does it get the data for the second parameter?**

The second parameter is retrieved from DLL3.dll. It is the structure pointer to AT\_INFO structure.

**7. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following: DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?**



**8. How can you load DLL2.dll into IDA Pro so that it matches the load address used by OllyDbg?**

We can change the base address at IDA Pro. Each time we load the DLL file we have never changed the new image base address. We can change the address to 0x320000.

