Lab 4 – Solutions
Name: Candice Renee Romeo Gomes
ID: 1001427308

ae8a8530a53c91a0c330445fe8d8297c

1.
   a. Main is located at 0x004011A0
   b. Main function looks for an active internet connection and then tries to make a request to http://reversing.rocks
   c. We see if else statement with a possible break in the loop.
   d. Interesting strings:
      - http://reversing.rocks
      - text/html
2.
   a. InternetConnectA has the following arguments:
      - hInternet – handle returned by a earlier call to InternetOpen
      - szServerName – IP address or name of the host server. In this case, "reversing.rocks"
      - nServerPort – Port number for the service – 1234.
      - lpszUserName – The name of the user logged on. 0 mean it is an anonymous user
      - lpszPassword
      - dwService - Type of service to access (FTP, Gopher, HTTP)
      - dwFlags
      - dwContext

   b. The function is making an HTTP request to server – http://reversing.rocks at port 1234.

   c. There are two if else statements in the function:

```
if(…){
        if(…){
        }
        else{
        }
else{
}
```

3.
   a. We see 3 while loops and 2 if else statements
   b. Imported function calls:
      - InternetWriteFile
      - HttpSendRequestExA
      - HttpOpenRequestA
      - FindFirstFileA
      - FindNextFileA
      - HttpEndRequestA
      - InternetCloseHandle
      - FindClose
   c. The subroutine sub_ 00401000 is trying to locate a file on the system. If the file is found, the malware will exit. However, if that file is not found we see the malware making HTTP post requests to write data to an open internet file.

4.
The malware checks for active internet connection and access http://reversing.rocks. Here it will try to access a file or write some sensitive data stored on the system to an open internet file at that server using HttpSendRequestA.

e1250254abbbeee84e04d9adcee31e63

1.

a. Imported functions:
   - AllocConsole
   - FindWindowA
   - ShowWindow
   - Fopen
   - Time
   - Fputs
   - Ctime
   - Fclose

b. Properties:
   - AllocConsole – allocates new console for the calling process
   - FindWindowA – retrieves a handle of the window requested to the calling process.
   - ShowWindow – shows the specified window
   - Fopen – opens a file
   - Time – retrieves the time of the system and the individual files.
   - Fputs – copies the characters from a specified address until it reaches null.
   - Ctime – converts time into a calendar local time and then textual representation
   - Fclose – closes an open file

c. Interesting strings:
   - \\WINDOWS\\lzwindowlz.av
   - ConsoleWindowClass
   - \nStarted logging:

2.

a. Imported functions called:
   - GetAsyncKeyState
   - Fopen

b. Switch statement with a jump table.

3.

Because of strings like "\r\n[SHIFT]\r\n" and "fopen" we can assume that this malware is a keylogger. Because of other strings like unknown-g@inbox.com we can assume that the malware wants to create an email and log information about the keys the user pushes and send that email to unknown email account.

4.

Useful signatures:

- helo typical-jam2.0catch.com\n
- unknown-g@hotmail.co.uk
- unknown-g@inbox.com
- my.inbox.com
- Subject:
- Logged

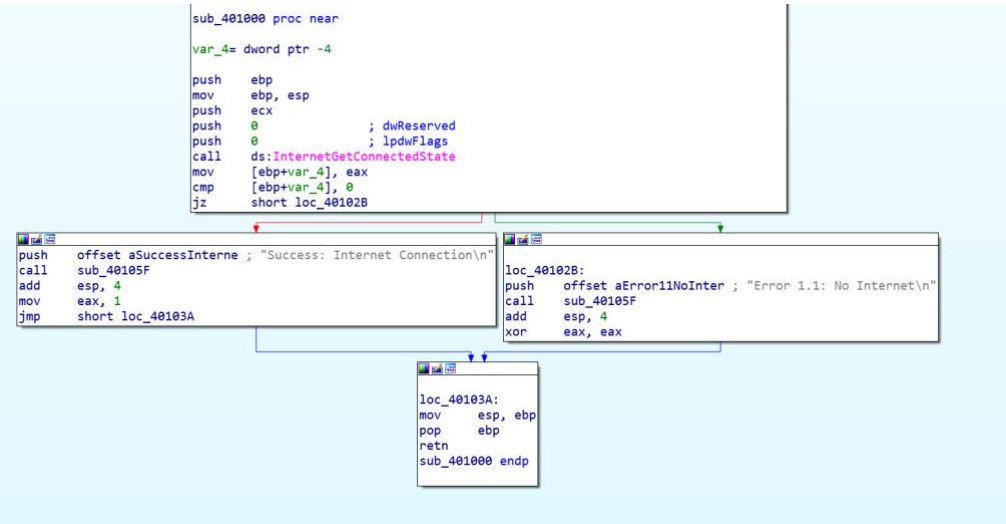
These signatures are useful because they tell us how the malware can be used as a keylogger and to identify the attacker.

5.

Yes, the malware creates a TIL file, where it writes information about the system and the user. This file can them be emailed to one of the email addresses in the previous solution.

6abde2f83015f066385d27cff6143c44

1.



Major code construct is an if statement found at 0x401000

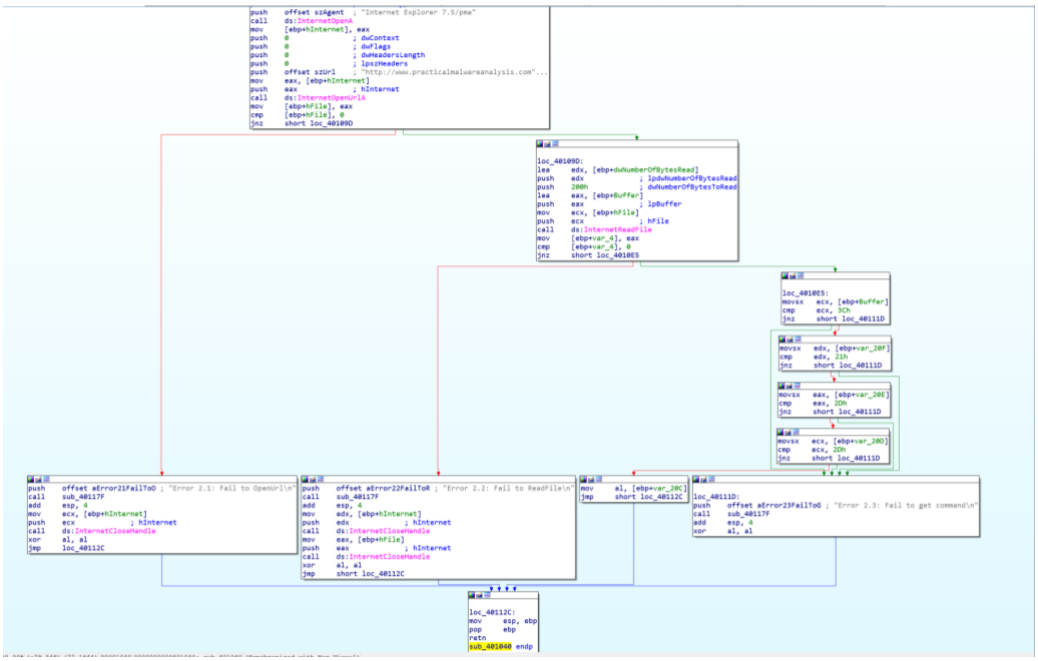
2.
The subroutine is called twice in the malware. We can find this by hitting the X key after selecting subroutine sub_40105F at 0x40105F location. Right before the subroutine is called we see strings being pushed to the stack. Namely, "Success: Internet Connection\n" and "Error 1.1: No Internet\n". Therefore, we can assume that the printf function is being called at this subroutine.

3.
Not much can be found using Static and Dynamic analysis. However, from the imported DLLs, I found WINNET.dll.InternetGetConnectedState. Using this and the strings found in the above question, we can assume that the malware is trying to get if the system has an active internet connection. It prints out Success or Error depending on the results it finds. It can also be used to check the status before establishing a connection with the internet.

c0b54534e188e1392f28d17faff3d454

1.
The first subroutine is at 0x401000. We see the same graph as the one in solution of malware 6abde2f83015f066385d27cff6143c44, question 1. We see an if function where the malware prints out "Success: Internet Connection\n" and "Error 1.1: No Internet\n" after checking for active internet connection.

2.
The subroutine at location 0x0040117F is sub_40117F. However, here we see it trying to print out the following strings:
"Success: Internet Connection\n"
"Error 1.1: No Internet\n"
"Error 2.1: Fail to OpenUrl\n"
"Error 2.2: Fail to ReadFile\n"
"Error 2.3: Fail to get command\n"
"Success: Parsed command is %c\n"



3.

The second subroutine is located at 0x401040. Here we see that the malware makes connection with 'http://www.practicalmalwareanalysis.com/cc.htm' using Internet Explorer 7.5/pma. We see that it tries compare the start of the file with 3Ch, 21h, 2Dh and 2Dh. When we convert it into ascii we get “<!--” which is the start of a comment on a HTML file. Therefore, we can say that the malware tries to read a comment of a HTML file at 'http://www.practicalmalwareanalysis.com/cc.htm’.

4.

The second subroutine is a series of if nested if else statements as below:

```
if(…){
        if(…){
                if(…){
                }
                else break;
                if(…){
                }
                else break;
                if(…){
                }
                else break;
                if(…){
                        read file;
                }
                else{
                }
        else{
        }
else{
}
```

First it compares if it the malware can open a connection with 'http://www.practicalmalwareanalysis.com/cc.htm. It then checks if the malware can “InternetReadFile”. The malware will then compare the file byte by byte with the results it obtains from “InternetReadFile” to see if the file is the html file it was looking for.

5.
Network based signatures:
http://www.practicalmalwareanalysis.com/cc.htm
Internet Explorer 7.5/pma

6.
The malware will first check for an active internet connection. If this is established it opens up the Internet Explorer 7.5 and contacts http://www.practicalmalwareanalysis.com/cc.htm. From here it accesses a HTML page. This page starts with a HTML comment "<!--". It then parses the strings from the html page one character at a time and prints out "Success: Parsed command is %c\n", where %c is the character parsed.

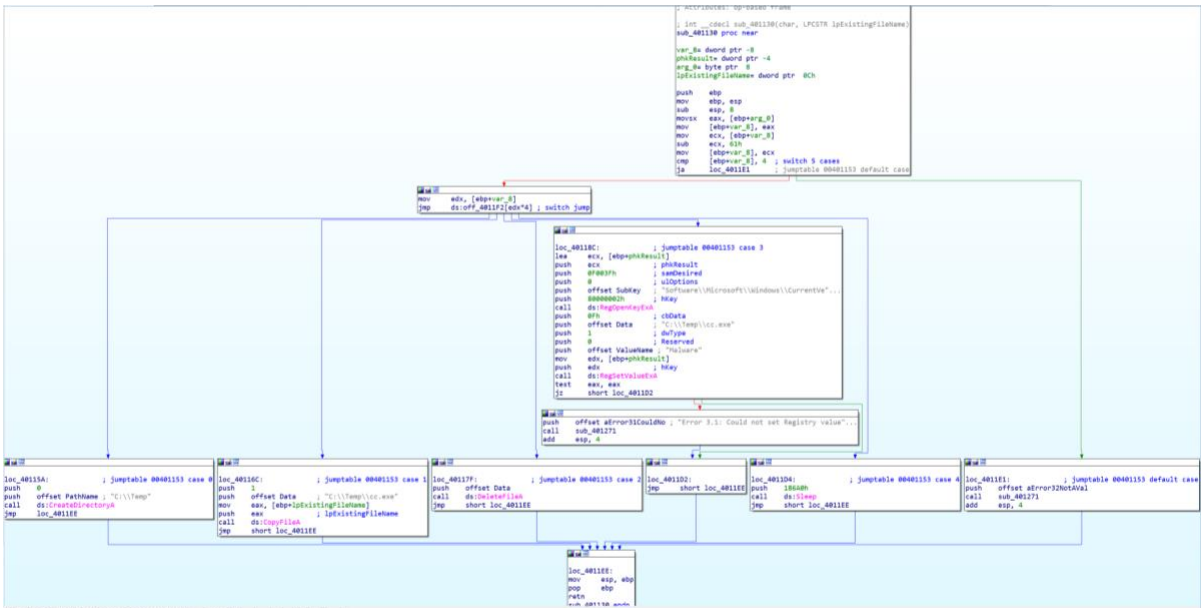
3f8e2b945deba235fa4888682bd0d640

1.
The main in this malware is similar to the previous malware. We see the malware looking for active internet connection and printing the success or error statement. We also see the access to html page at http://www.practicalmalwareanalysis.com/cc.htm. However, the addition is a subroutine sub_401130.

2.
There are 2 arguments for the new function:

- arg_0 - The bytes in for the existing file name.
- var_8 – The character parsed from the HTML file.

3.



From the figure above we can clearly see that this function is a switch statement with a jump table, for 5 switch cases and one default case.

4.
From the assembly calls we can say that the function can:

- set a registry value,
- create a directory,
- copy a file,

- delete a file,
- sleep for 100000 milliseconds or
- print an error message

5.
Host based signatures:
- Software\Microsoft\Windows\CurrentVersion\Run\Malware
- C:\Temp\cc.exe
- Malware

6.
The malware is going to check for an active internet connection, then it will open a web browser and connect to http://www.practicalmalwareanalysis.com/cc.htm to access the comment of an HTML page. Depending on the first character of this comment, the malware will use a swtch case statement to either
- set a registry value,
- create a directory,
- copy a file,
- delete a file,
- sleep for 100000 milliseconds or
- print an error message

21be74dfafdacaaab1c8d836e2186a69

1.
Calls from main method:
- At 0x401000 the malware checks for an active internet connection.
- At 0x401040 the malware parses the HTML comment,
- At 0x401150 is we see a switch case statement.
- At 0x4012B5 is we see the call to printf statement.

2.

```
.text:00401248 ; ---------------------------------------------------------------------------
.text:00401248
.text:00401248 loc_401248:                             ; CODE XREF: _main+12↑j
.text:00401248                 mov     [ebp+var_C], 0
.text:0040124F                 jmp     short loc_40125A
.text:00401251 ; ---------------------------------------------------------------------------
.text:00401251
.text:00401251 loc_401251:                             ; CODE XREF: _main+7D↓j
.text:00401251                 mov     eax, [ebp+var_C]
.text:00401254                 add     eax, 1
.text:00401257                 mov     [ebp+var_C], eax
.text:0040125A
.text:0040125A loc_40125A:                             ; CODE XREF: _main+1F↑j
.text:0040125A                 cmp     [ebp+var_C], 5A0h
.text:00401261                 jge     short loc_4012AF
.text:00401263                 mov     ecx, [ebp+var_C]
.text:00401266                 push    ecx
.text:00401267                 call    sub_401040
```

From the figure above, we can see that a for loop has been added to the main method. We can see the at a variable is being initialized to zero and counter being incremented by 1. We also see a jump (loop) to the incrementing location.

3.
The function at subroutine sub_401040 parses the HTML comment. We see that it takes a parameter (counter variable) and calls sprintf to print string “Internet Explorer 7.50/pma%d”. %d is replaced with the counter and the string is passed to InternetOpenA. This is can be used to represent different a User-Agent during the HTTP communication as the counter increases.

4.
Because of the for loop, the malware will sleep for 60 seconds and it will repeat this process 1440 times. Therefore, the computer will sleep for 1440 minutes which is 24 hours.

5.
Network based signatures

- Internet Explorer 7.50/pma%d
- http://www.practicalmalwareanalysis.com/cc.htm

6.
The program check for an active internet connection. It prints out success or error depending on the result. It uses Internet Explorer 7.50/pma%d to send an HTTP request which consists of a unique User-Agent to access a HTML page: http://www.practicalmalwareanalysis.com/cc.htm. This web page tracks how long the program has been running using the counter. The accessed webpage will contain a comment. Depending on this comment, the malware will:

- set a registry value,
- create a directory,
- copy a file,
- delete a file,
- sleep for 100000 milliseconds or
- print an error message

The malware runs for 1440 minutes.

c04fd8d9198095192e7d55345966da2e

1.
The malware creates a service called MalService in the mail and makes a call StartServiceCtrlDispatcherA which Connects MalService to the service control manager, and make changes like starting the malware to achieve persistence.

2.
We see 2 call to mutex related functions, first is OpenMutexA, which is looking to obtain a handle of a mutex named HGL345 and the second is CreateMutexA, to create a mutex named HGL345. If OpenMutex would have found a handle to HGL345 the mutex will not be created and ExitProcess function will be called. This tells us that, the malware is designed to ensure that only one copy of this executable is running on a system at any given time.

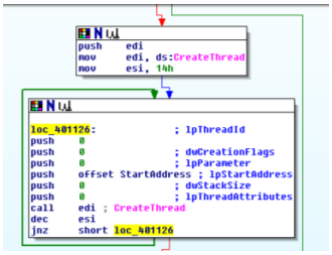
3.
Host base signatures:

- MalService
- HGL345

4.
Network based signatures:

- Internet Explorer 8.0
- www.malwareanalysisbook.com

5.



We see that the malware tries to create threads using CreateThread. This is within the loop 20 times (14h), which leads us to believe that it tries to create 20 threads. From the arguments of CreateThread we see that it has a start

address for creating threads. Proceeding this we see function calls to InternetOpen and InternetOpenUrlA where it tries to access "https://www.malwareanalysisbook.com".

From the above images, we can assume that the malware intends to install itself on several machines and enable a DDoS attack on https://www.malwareanalysisbook.com.

6.

Here we see API calls to SystemTimeToFileTime, CreateWaitableTimerA, SetWaitableTimer and WaitForSingleObject. Just before this we see the malware setting the variables for year, dayofweek, hour and second to zero. The malware then sets Year to 834h which is 2100. Therefore, we can assume that the malware is set to wait unit midnight January 1st, 2100 to terminate.

7bbc691f7e87f0986a1030785268f190

1.
We do not see any evidence of the malware trying to achieve persistence.

2.
After running the malware, we see that it opens a web browser and accesses a webpage at: http://www.malwareanalysisbook.com/ad.html.

3.
The malware terminates as soon as the advertisement on the webpage is executed.

bd62dab79881bc6ec0f6be4eef1075bc & 290934c61de9176ad682ffdd65f0a669

1.
This program achieves persistence by writing a new DLL, kerne132 to C:\Windows\ System32. It uses this to modifying every .exe file on the system. Every .exe file on the system will now import this dll.

2.
Host based signatures:

- C:\windows\system32\kerne132.dll
- WARNING_THIS_WILL_DESTROY_YOUR_MACHINE
- .exe
- Lab07-03.dll
- SADFHUHF

3.
In the dll we see strings hello, 127.26.152.13 and sleep which tell us that the malware establishes a backdoor connection with a remote host where the attacker can execute commands using CommandLine and shutdown the system.

4.
To get rid of the malware:

- We could use a backup of the system
- We could get a new copy of kernel32.dll and name it as kerne123.dll
- We could write a program that will undo the changes to the .exe files