Lab 12-01

Analyze the malware found in the file Lab12-01.exe and Lab12-01.dll.

Make sure that these files are in the same directory when performing the analysis.

This chapter was about all forms of performing the code in presence of other process, there were mentioned methods such as injecting the library, process hollowing, hooking the procedures and executing the code through APC.

In this lab we were given two files, an executable and dynamic library link file.

Let's start off with simple skimming the files with basic statis analysis to see with what we might be dealing with. At the first sight the given files seem not packed, nor obfuscated.

Interesting strings found in both files:

- Press OK to reboot
- Practical Malware Analysis %d

The executable is capable of:

- Injecting memory into any process
- Dynamically loading libraries and its functions
- Allocating data on heap
- File operations
- Managing threads

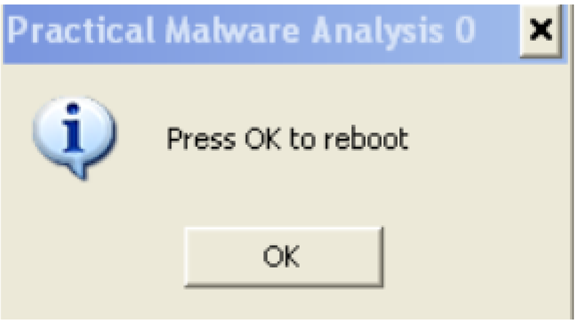
The dynamic link library file is capable of:

- Managing threads
- Managing processes
- Dynamic loading libraries and its functions
- Retrieving thread's last error code value
- Heap management
- File operations

The basic static analysis doesn't tell much, let's run the file to see what it does.

When it comes to the malware activity, the process monitor shows that the file tries to locate psiapi.dll, and when it's found it uses filemapping to retrieve its data. The original file remains untouched (same hash). Quickly after that – the process terminates.

As soon as you run it, it displays the messagebox provided below.


Practical Malware Analysis 0

Press OK to reboot

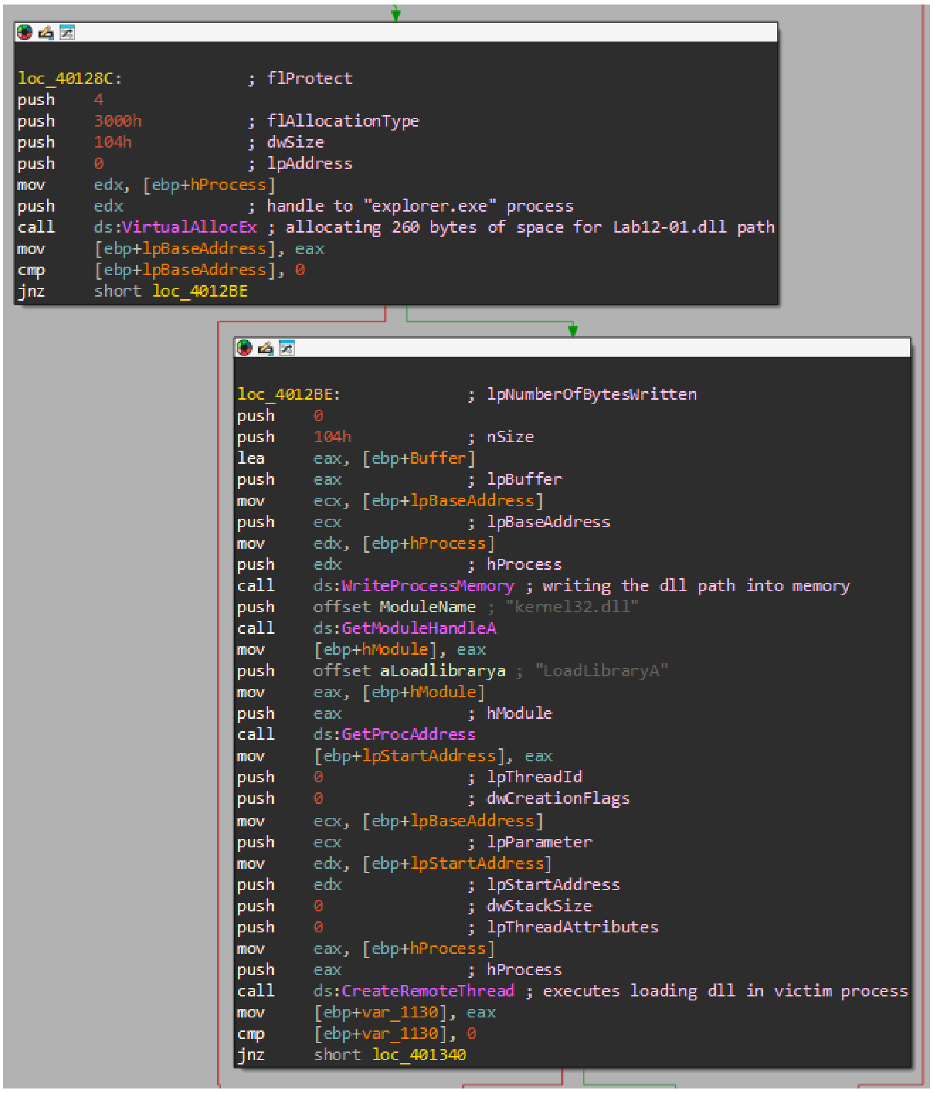
When you click okay, the second message box appear with incrementing value by 1 at the end of the MessageBox title.

Let's drop the files into IDA and Olly.

Executable is intended to dynamically load psapi.dll, a library used for process/drivers listing, in this scenario malware wants to get a list of all currently running processes.

To do that, it loads three libraries for further usage: **GetModuleBaseName**, **EnumProcess**, **EnumProcessModule**.

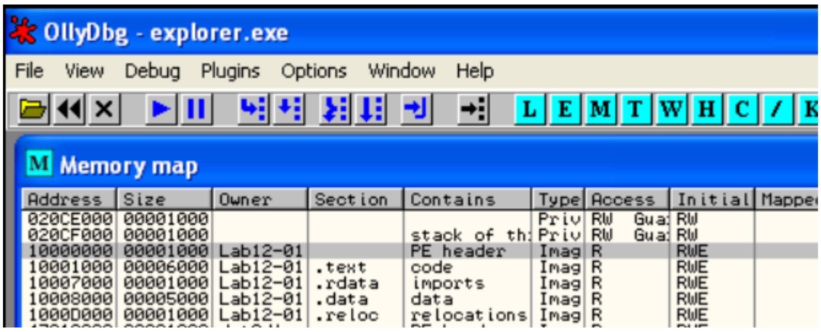
After retrieving the processes, it iterates over each process in order to find "explorer.exe", then it performs an dll injection attack. It tries to load "Lab12-01.dll" into the victim process.



```
loc_40128C:              ; flProtect
push    4
push    3000h            ; flAllocationType
push    104h             ; dwSize
push    0                ; lpAddress
mov     edx, [ebp+hProcess]
push    edx              ; handle to "explorer.exe" process
call    ds:VirtualAllocEx ; allocating 260 bytes of space for Lab12-01.dll path
mov     [ebp+lpBaseAddress], eax
cmp     [ebp+lpBaseAddress], 0
jnz     short loc_4012BE
```

```
loc_4012BE:              ; lpNumberOfBytesWritten
push    0
push    104h             ; nSize
lea     eax, [ebp+Buffer]
push    eax              ; lpBuffer
mov     ecx, [ebp+lpBaseAddress]
push    ecx              ; lpBaseAddress
mov     edx, [ebp+hProcess]
push    edx              ; hProcess
call    ds:WriteProcessMemory ; writing the dll path into memory
push    offset ModuleName ; "kernel32.dll"
call    ds:GetModuleHandleA
mov     [ebp+hModule], eax
push    offset aLoadlibrarya ; "LoadLibraryA"
mov     eax, [ebp+hModule]
push    eax              ; hModule
call    ds:GetProcAddress
mov     [ebp+lpStartAddress], eax
push    0                ; lpThreadId
push    0                ; dwCreationFlags
mov     ecx, [ebp+lpBaseAddress]
push    ecx              ; lpParameter
mov     edx, [ebp+lpStartAddress]
push    edx              ; lpStartAddress
push    0                ; dwStackSize
push    0                ; lpThreadAttributes
mov     eax, [ebp+hProcess]
push    eax              ; hProcess
call    ds:CreateRemoteThread ; executes loading dll in victim process
mov     [ebp+var_1130], eax
cmp     [ebp+var_1130], 0
jnz     short loc_401340
```

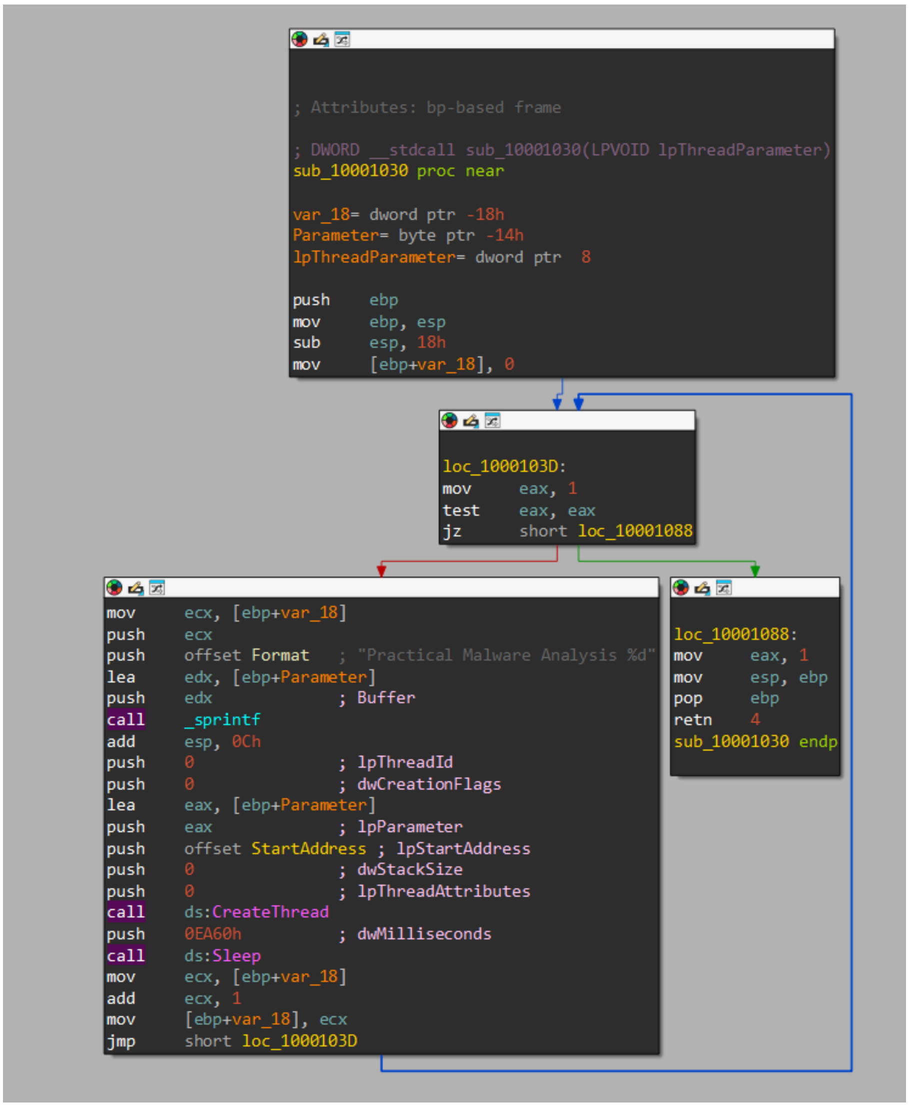
To confirm that, I attached the debugger to the attacked process, and at the memory map there is clearly visible injected malicious dll.



```
Address  Size     Owner    Section Contains     Type Access    Initial Mappe
020CE000 00001000                               Priv RW   Gua: RW
020CF000 00001000                  stack of th: Priv RW   Gua: RW
10000000 00001000 Lab12-01         PE header    Imag R         RWE
10001000 00006000 Lab12-01 .text   code         Imag R         RWE
10007000 00001000 Lab12-01 .rdata  imports      Imag R         RWE
10008000 00005000 Lab12-01 .data   data         Imag R         RWE
1000D000 00001000 Lab12-01 .reloc  relocations  Imag R         RWE
```

Now, let’s check what the dll does in IDA.

The main code check if the dll is being attached, if it does – then it calls **sub_10001030**.



```
; Attributes: bp-based frame

; DWORD __stdcall sub_10001030(LPVOID lpThreadParameter)
sub_10001030 proc near

var_18= dword ptr -18h
Parameter= byte ptr -14h
lpThreadParameter= dword ptr  8

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp+var_18], 0
```

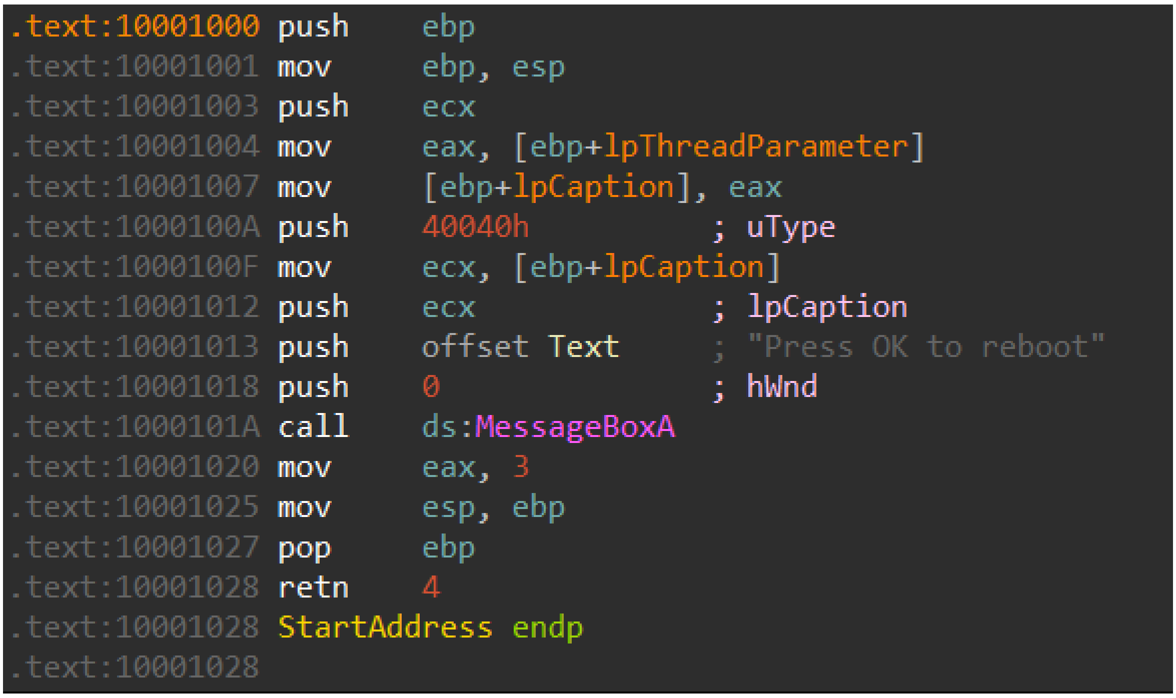
```
loc_1000103D:
mov     eax, 1
test    eax, eax
jz      short loc_10001088
```

```
mov     ecx, [ebp+var_18]
push    ecx
push    offset Format   ; "Practical Malware Analysis %d"
lea     edx, [ebp+Parameter]
push    edx             ; Buffer
call    _sprintf
add     esp, 0Ch
push    0               ; lpThreadId
push    0               ; dwCreationFlags
lea     eax, [ebp+Parameter]
push    eax             ; lpParameter
push    offset StartAddress ; lpStartAddress
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
call    ds:CreateThread
push    0EA60h          ; dwMilliseconds
call    ds:Sleep
mov     ecx, [ebp+var_18]
add     ecx, 1
mov     [ebp+var_18], ecx
jmp     short loc_1000103D
```

```
loc_10001088:
mov     eax, 1
mov     esp, ebp
pop     ebp
retn    4
sub_10001030 endp
```

Here, we have a string containing “Practical Malware Analysis %d”, a buffer counting created threads, a startaddress pointing to sub_10001000, CreateThread call, and after the return there is a Sleeping function for 60 seconds, and all that is performed in an infinite while (eax>0) loop.

Sub_10001000 is popping a MessageBox on the screen with incrementing value of the created threads every 60 seconds (sleep method after CreateThread)

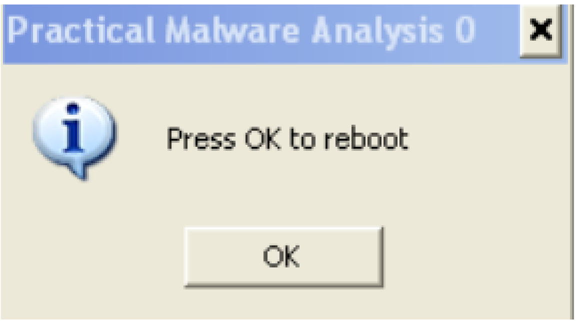


```
.text:10001000 push    ebp
.text:10001001 mov     ebp, esp
.text:10001003 push    ecx
.text:10001004 mov     eax, [ebp+lpThreadParameter]
.text:10001007 mov     [ebp+lpCaption], eax
.text:1000100A push    40040h          ; uType
.text:1000100F mov     ecx, [ebp+lpCaption]
.text:10001012 push    ecx             ; lpCaption
.text:10001013 push    offset Text     ; "Press OK to reboot"
.text:10001018 push    0               ; hWnd
.text:1000101A call    ds:MessageBoxA
.text:10001020 mov     eax, 3
.text:10001025 mov     esp, ebp
.text:10001027 pop     ebp
.text:10001028 retn    4
.text:10001028 StartAddress endp
.text:10001028
```

Questions

1. What happens when you run the malware executable?

When we run the executable, we immediately see a MessageBox window popping out. When we click “OK” nothing visible happens.



Practical Malware Analysis 0

Press OK to reboot

2. What process is being injected?

The victim process that is being injected by the “Lab12-01.dll” is “explorer.exe”.

3. How can you make the malware stop the pop-ups?

To stop the pop-ups we have to restart explorer.exe or kill the main thread of the injected malicious dll.

The malware does not perform any action towards achieving persistence, so a simple reboot will do the thing too.

4. How does this malware operate?

Malware perform DLL injection “Lab12-01.dll” into explorer.exe process in order to execute malicious activity. When it does that, it show message boxes with a counter at the end of the window title.