Lab 09-01

Analyze the malware found in the file Lab09-01.exe using OllyDbg and IDA Pro to answer the following questions.

This malware was initially analyzed in the Chapter 3 labs using basic static and dynamic analysis techniques.

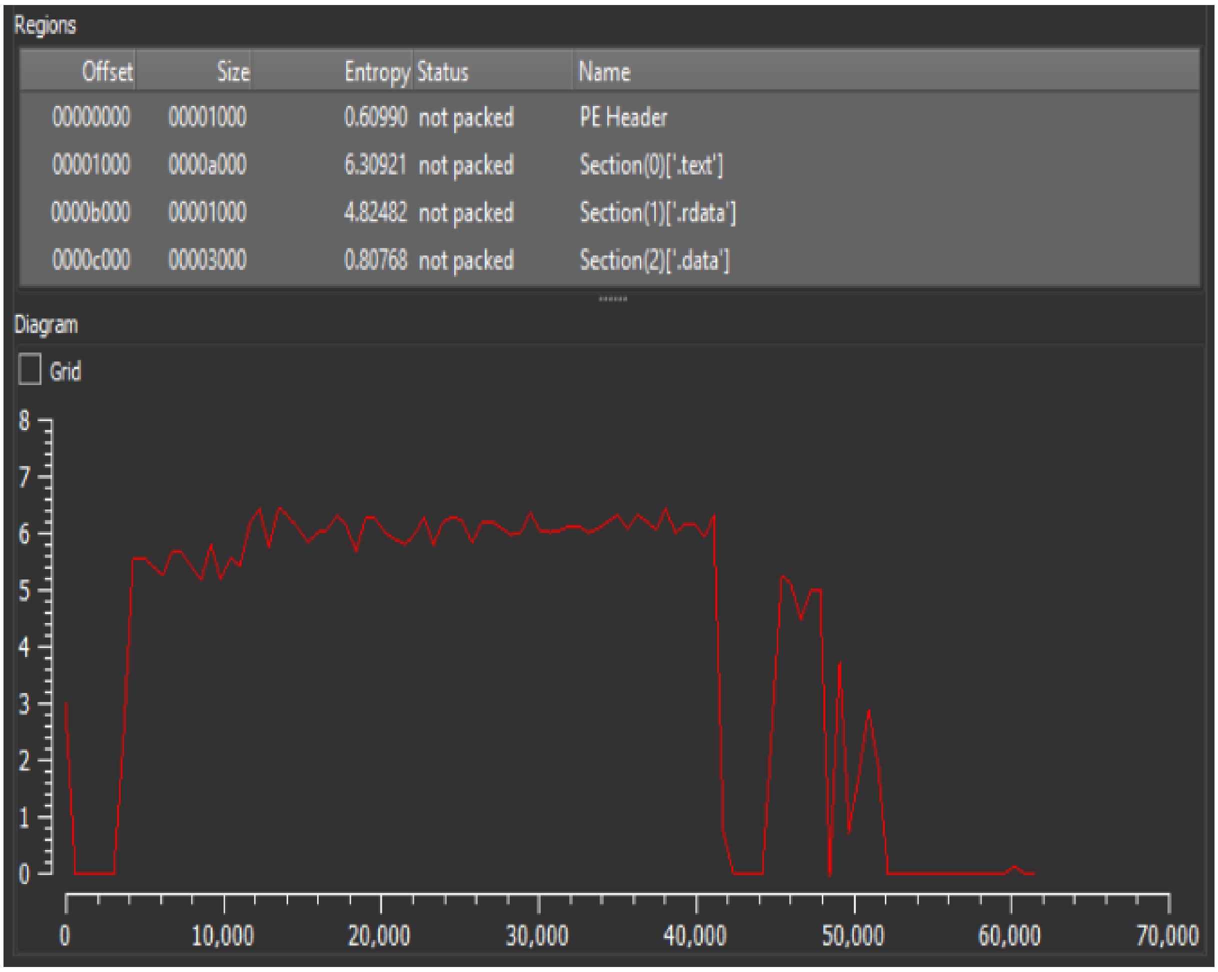
Contents of Lab 09-01 analysis

1. Preliminary Analysis

First of all what I'm going to do is to check whether the file is packed or obfuscated.

Raw sizes of physical and memory are almost the same, libraries are visible and section names are normal.

The entropy seems to be in range, there's a higher value at .text, but nothing unusual, we will verify that later on.



Regions

Offset	Size	Entropy	Status	Name
00000000	00001000	0.60990	not packed	PE Header
00001000	0000a000	6.30921	not packed	Section(0)['.text']
0000b000	00001000	4.82482	not packed	Section(1)['.rdata']
0000c000	00003000	0.80768	not packed	Section(2)['.data']

Strings found:

- %SYSTEMROOT%\system32\
- k:%s h:%s p:%s per:%s
- Manager Service
- NOTHING
- DOWNLOAD
- UPLOAD
- SLEEP

- command.com
- http://www.practicalmalwareanalysis.com

Imported libraries:

- KERNEL32.dll
- ADVAPI32.dll
- SHELL32.dll
- WS2_32.dll

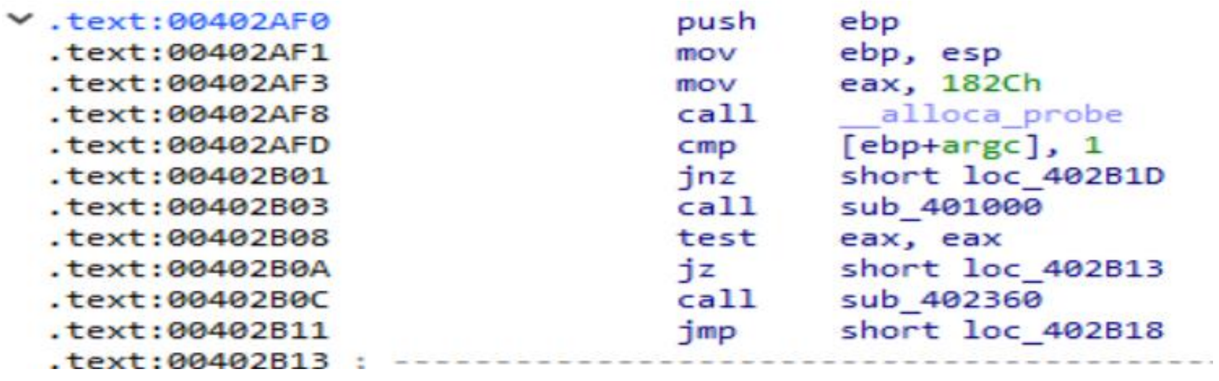
There are a lot of functions defined, I wonder if actually all of them are used, or they were just faked to mislead an analyst.

Based on the functions it has themes of:

- File System Operations
  - Copy file
  - Write file
  - Read file
  - Create file
  - Directory operations
  - Reading system directory
- Process Operations
  - Creating process
- Network operations
  - C2 connection (socket, send, connect, gethostbyname)
- Shell operations
  - Executing shellcode
- Registry operations
  - Add, modify and delete values
  - Create keys
  - Read key values
- Services operations
  - Create services
  - Delete services
- Dynamical library loading
  - Load libraries during run-time
  - Gather memory address of library functions

2. IDA & x32dbg Analysis

After loading the file into IDA, the first thing we notice is checking for argc passed parameters number at 0x402AFD.

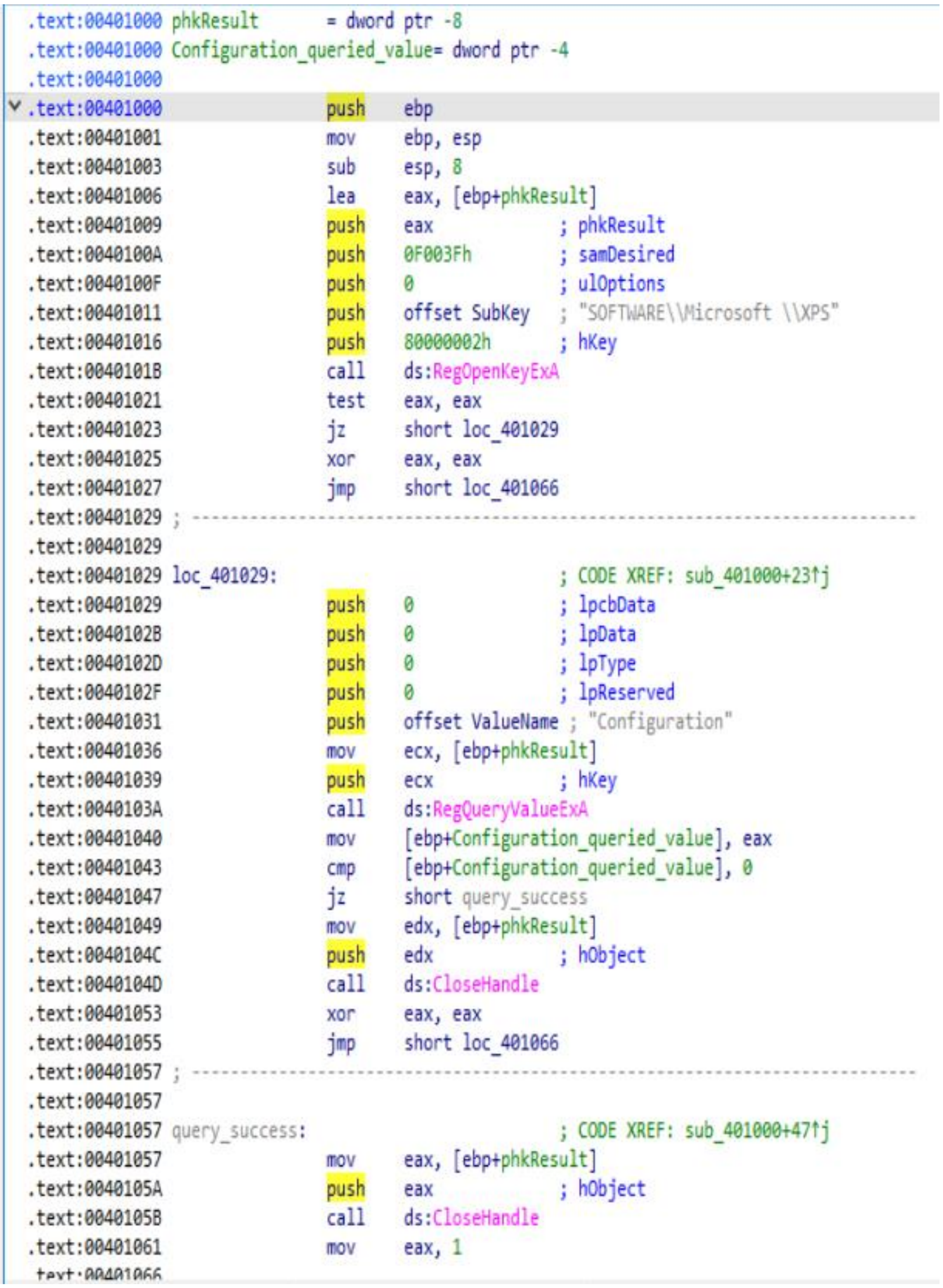


```
.text:00402AF0                 push    ebp
.text:00402AF1                 mov     ebp, esp
.text:00402AF3                 mov     eax, 182Ch
.text:00402AF8                 call    __alloca_probe
.text:00402AFD                 cmp     [ebp+argc], 1
.text:00402B01                 jnz     short loc_402B1D
.text:00402B03                 call    sub_401000
.text:00402B08                 test    eax, eax
.text:00402B0A                 jz      short loc_402B13
.text:00402B0C                 call    sub_402360
.text:00402B11                 jmp     short loc_402B18
.text:00402B13 ; ---------------------------------------
```

From this step, we have two ways of following the program – we either pass only executing parameter, or provide additional ones.

2.1. First Path (1 parameter)

If we provide only one parameter, then we move **sub_401000**.



```
.text:00401000 phkResult       = dword ptr -8
.text:00401000 Configuration_queried_value= dword ptr -4
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 8
.text:00401006                 lea     eax, [ebp+phkResult]
.text:00401009                 push    eax             ; phkResult
.text:0040100A                 push    0F003Fh         ; samDesired
.text:0040100F                 push    0               ; ulOptions
.text:00401011                 push    offset SubKey   ; "SOFTWARE\\Microsoft \\XPS"
.text:00401016                 push    80000002h       ; hKey
.text:0040101B                 call    ds:RegOpenKeyExA
.text:00401021                 test    eax, eax
.text:00401023                 jz      short loc_401029
.text:00401025                 xor     eax, eax
.text:00401027                 jmp     short loc_401066
.text:00401029 ; ---------------------------------------------------------------------------
.text:00401029
.text:00401029 loc_401029:                             ; CODE XREF: sub_401000+23↑j
.text:00401029                 push    0               ; lpcbData
.text:0040102B                 push    0               ; lpData
.text:0040102D                 push    0               ; lpType
.text:0040102F                 push    0               ; lpReserved
.text:00401031                 push    offset ValueName ; "Configuration"
.text:00401036                 mov     ecx, [ebp+phkResult]
.text:00401039                 push    ecx             ; hKey
.text:0040103A                 call    ds:RegQueryValueExA
.text:00401040                 mov     [ebp+Configuration_queried_value], eax
.text:00401043                 cmp     [ebp+Configuration_queried_value], 0
.text:00401047                 jz      short query_success
.text:00401049                 mov     edx, [ebp+phkResult]
.text:0040104C                 push    edx             ; hObject
.text:0040104D                 call    ds:CloseHandle
.text:00401053                 xor     eax, eax
.text:00401055                 jmp     short loc_401066
.text:00401057 ; ---------------------------------------------------------------------------
.text:00401057
.text:00401057 query_success:                          ; CODE XREF: sub_401000+47↑j
.text:00401057                 mov     eax, [ebp+phkResult]
.text:0040105A                 push    eax             ; hObject
.text:0040105B                 call    ds:CloseHandle
.text:00401061                 mov     eax, 1
```

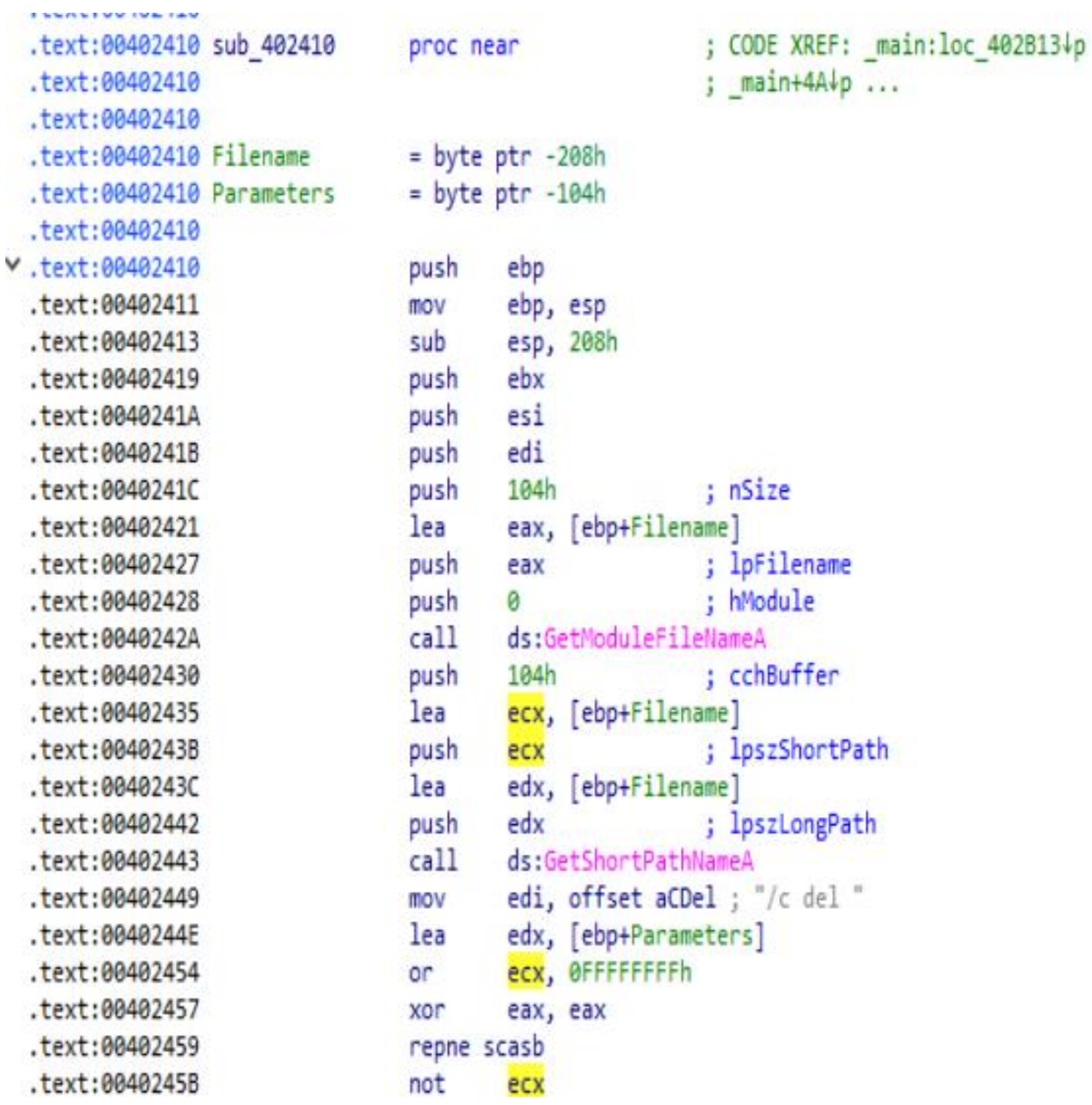
Sub_401000 is intended to verify if there is registry subkey located at "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\XPS".

If it doesn't exist – we move to sub_402410.

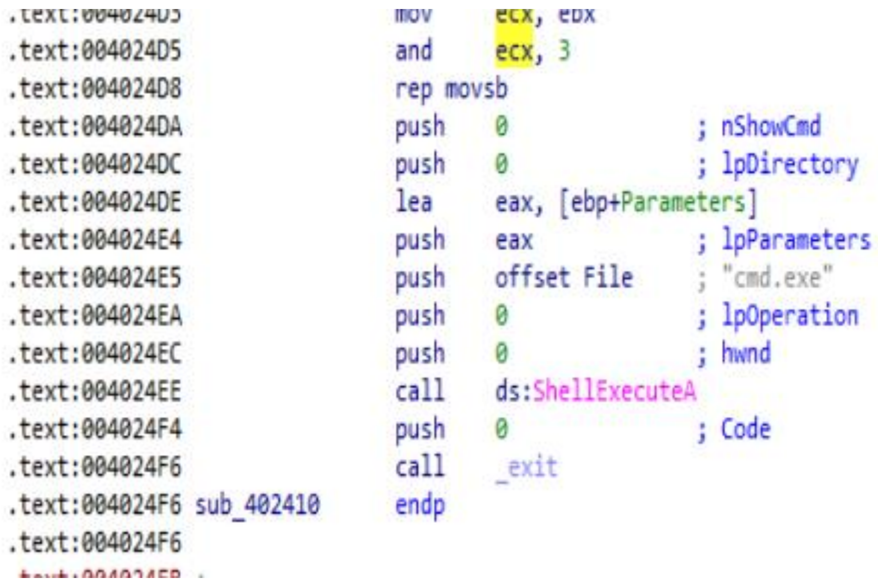
If it exists – we move to sub_402360.

2.1.1. First path (1 parameter, registry not existing)

After registry check for the subkey results in an error, we move to **sub_402410**.



```
.text:00402410 sub_402410      proc near               ; CODE XREF: _main:loc_402B13↓p
.text:00402410                                         ; _main+4A↓p ...
.text:00402410
.text:00402410 Filename        = byte ptr -208h
.text:00402410 Parameters      = byte ptr -104h
.text:00402410
.text:00402410                 push    ebp
.text:00402411                 mov     ebp, esp
.text:00402413                 sub     esp, 208h
.text:00402419                 push    ebx
.text:0040241A                 push    esi
.text:0040241B                 push    edi
.text:0040241C                 push    104h            ; nSize
.text:00402421                 lea     eax, [ebp+Filename]
.text:00402427                 push    eax             ; lpFilename
.text:00402428                 push    0               ; hModule
.text:0040242A                 call    ds:GetModuleFileNameA
.text:00402430                 push    104h            ; cchBuffer
.text:00402435                 lea     ecx, [ebp+Filename]
.text:0040243B                 push    ecx             ; lpszShortPath
.text:0040243C                 lea     edx, [ebp+Filename]
.text:00402442                 push    edx             ; lpszLongPath
.text:00402443                 call    ds:GetShortPathNameA
.text:00402449                 mov     edi, offset aCDel ; "/c del "
.text:0040244E                 lea     edx, [ebp+Parameters]
.text:00402454                 or      ecx, 0FFFFFFFFh
.text:00402457                 xor     eax, eax
.text:00402459                 repne scasb
.text:0040245B                 not     ecx
```

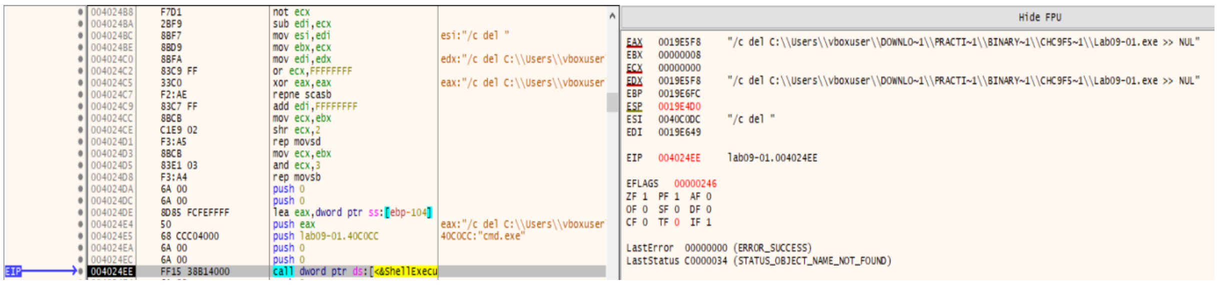


```
.text:004024D3                 mov     ecx, ebx
.text:004024D5                 and     ecx, 3
.text:004024D8                 rep movsb
.text:004024DA                 push    0               ; nShowCmd
.text:004024DC                 push    0               ; lpDirectory
.text:004024DE                 lea     eax, [ebp+Parameters]
.text:004024E4                 push    eax             ; lpParameters
.text:004024E5                 push    offset File     ; "cmd.exe"
.text:004024EA                 push    0               ; lpOperation
.text:004024EC                 push    0               ; hwnd
.text:004024EE                 call    ds:ShellExecuteA
.text:004024F4                 push    0               ; Code
.text:004024F6                 call    _exit
.text:004024F6 sub_402410      endp
.text:004024F6
```

Analyzing the provided code tell us that it gets a handle to a file, reads a filepath then uses ShellExecuteA to run the command cmd.exe /c del <file_path> >> NUL.

The nShowCmd parameter set to 0 makes sure that the cmd is not visible, and >> NUL at the end discards any output after the command.

In the debugger the command seems to delete its own file to evade detection.



```
 004024B8   F7D1          not ecx
 004024BA   2BF9          sub edi,ecx
 004024BC   8BF7          mov esi,edi                     esi:"/c del "
 004024BE   8BD9          mov ebx,ecx
 004024C0   8BFA          mov edi,edx                     edx:"/c del C:\\Users\\vboxuser
 004024C2   83C9 FF       or ecx,FFFFFFFF
 004024C5   33C0          xor eax,eax                     eax:"/c del C:\\Users\\vboxuser
 004024C7   F2:AE         repne scasb
 004024C9   83C7 FF       add edi,FFFFFFFF
 004024CC   8BCB          mov ecx,ebx
 004024CE   C1E9 02       shr ecx,2
 004024D1   F3:A5         rep movsd
 004024D3   8BCB          mov ecx,ebx
 004024D5   83E1 03       and ecx,3
 004024D8   F3:A4         rep movsb
 004024DA   6A 00         push 0
 004024DC   6A 00         push 0
 004024DE   8D85 FCFEFFFF lea eax,dword ptr ss:[ebp-104]
 004024E4   50            push eax                        eax:"/c del C:\\Users\\vboxuser
 004024E5   68 CCC04000   push lab09-01.40C0CC            40C0CC:"cmd.exe"
 004024EA   6A 00         push 0
 004024EC   6A 00         push 0
EIP 004024EE FF15 38B14000 call dword ptr ds:[<&ShellExecu
```

```
Hide FPU

EAX   0019E5F8    "/c del C:\\Users\\vboxuser\\DOWNLO~1\\PRACTI~1\\BINARY~1\\CHC9F5~1\\Lab09-01.exe >> NUL"
EBX   00000008
ECX   00000000
EDX   0019E5F8    "/c del C:\\Users\\vboxuser\\DOWNLO~1\\PRACTI~1\\BINARY~1\\CHC9F5~1\\Lab09-01.exe >> NUL"
EBP   0019E6FC
ESP   0019E4D0
ESI   0040C0DC    "/c del "
EDI   0019E649

EIP   004024EE    lab09-01.004024EE

EFLAGS   00000246
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
```

After the call, the application calls _exit and terminates.

2.1.2. First Path (1 parameter, registry existing)

When the registry check for the subkey results in a success, we move to sub_402360.

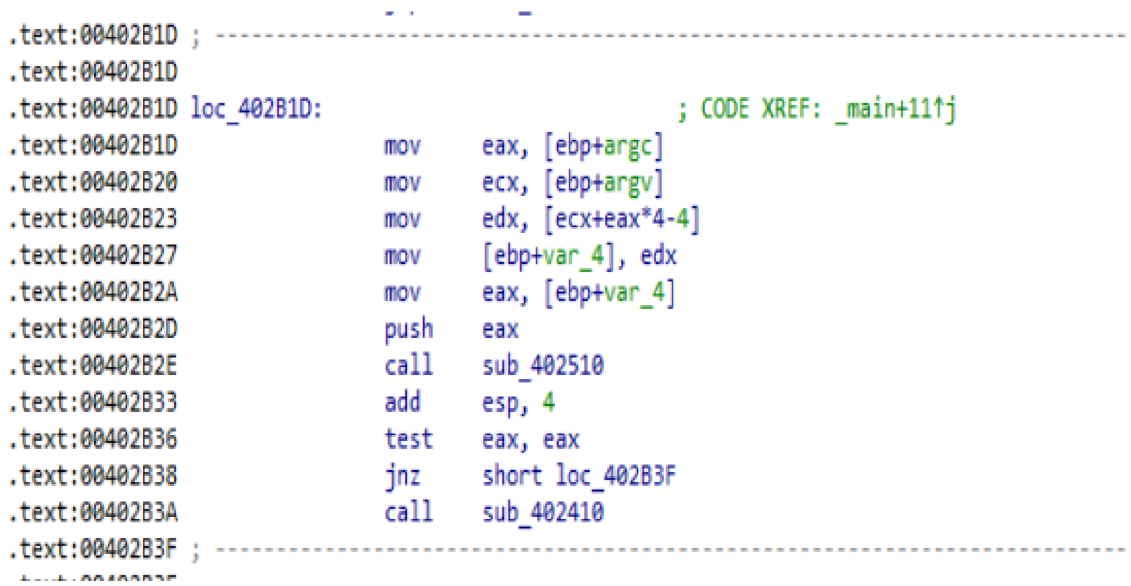
After a quick analysis I can tell that it opens the existing registry once again, modifies some data in it, then shows some network activity with C2.

We will come back to this at 2.4 Backdoor Analysis when we get to know more about this malware and its functions.

2.2. Providing parameter path (>2 parameters) & password verification

To run the malware and see its possibilities, we need to add atleast 1 parameter.

When we do that, we arrive at loc_402B1D.



```
.text:00402B1D ; ---------------------------------------------------------------------------
.text:00402B1D
.text:00402B1D loc_402B1D:                             ; CODE XREF: _main+11↑j
.text:00402B1D                 mov     eax, [ebp+argc]
.text:00402B20                 mov     ecx, [ebp+argv]
.text:00402B23                 mov     edx, [ecx+eax*4-4]
.text:00402B27                 mov     [ebp+var_4], edx
.text:00402B2A                 mov     eax, [ebp+var_4]
.text:00402B2D                 push    eax
.text:00402B2E                 call    sub_402510
.text:00402B33                 add     esp, 4
.text:00402B36                 test    eax, eax
.text:00402B38                 jnz     short loc_402B3F
.text:00402B3A                 call    sub_402410
.text:00402B3F ; ---------------------------------------------------------------------------
```

The first 6 lines of code are intended to load up onto the stack the last parameter passed, and call sub_402510.

Sub_402510 is intended to validate last parameter, something like password check.

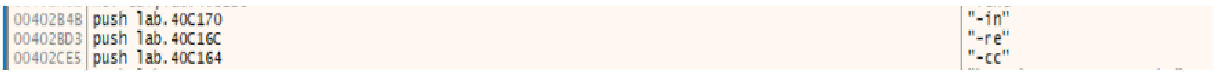
The password has to meet such requirements:

- Length of 4
- First letter – a
- Second letter – b
- Third letter – c
- Fourth letter – d

If we pass such string (abcd), then we continue with validation of the first argument passed.

2.3. Malware’s input parameters

During debugging the malware I found some interesting strings:



```
00402B4B | push lab.40C170        "-in"
00402BD3 | push lab.40C16C        "-re"
00402CE5 | push lab.40C164        "-cc"
```

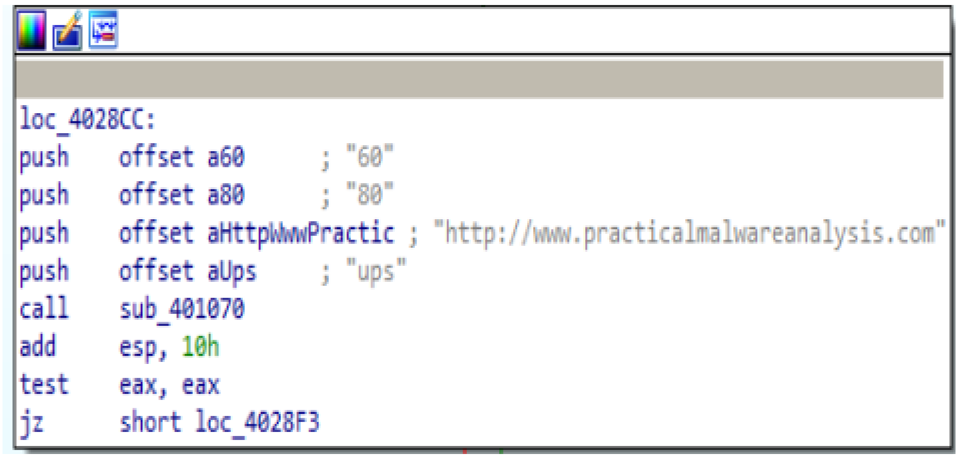
Those are arguments that change malware’s behavior. They are also visible in IDA.

2.3.1 -in parameter

When we provide -in parameter, and there are only total three parameters (file execution, -in parameter and the password “abcd”)

Here, we have several actions, I will list them in an successful order (without any errors)

- Tries to create service named as the executed file with appended string “Service Manager” with start type of 2 which is auto-start and lpBinaryPathName of "%SYSTEMROOT%\\system32\\Lab09-01.exe", if the service already exists, it refreshes the values and proceed to exit.
- Perform a CopyFile with its original file to "%SYSTEMROOT%\\system32\\ as I mentioned a line higher.
- Locates kernel32.dll in system directory, reads its data (LastWriteTime, LastAccessTime, CreationTime) and writes that data into its previously cloned file.
- Creates registry subkey XPS/Configuration and fill it with data (as shown on the picture below).



```
loc_4028CC:
push    offset a60      ; "60"
push    offset a80      ; "80"
push    offset aHttpWwwPractic ; "http://www.practicalmalwareanalysis.com"
push    offset aUps     ; "ups"
call    sub_401070
add     esp, 10h
test    eax, eax
jz      short loc_4028F3
```

Based on these actions we can tell that as the name of the parameter (-in) hints, providing this parameter “install” the malware on the OS which means it hid itself at system32, ensured that it stays active after reboot and utilized registry API to store some of the data (probably for further use).

2.3.2 -re parameter

When we dynamically run the malware with -re parameter while analyzing it with Process Monitor, we see that the malware makes the previous operations from -in, but in reverse.

First it deletes the service, deletes the file in system32 directory, deletes the XPS subkey in registry and terminates the process.

This parameter was intended to erase its footprints.

2.3.3 -c parameter

When the flag is applied, the malware performs a check to ensure that exactly 7 parameters are provided. For instance:

Lab09-01.exe -c par1 par2 par3 par4 abcd

If the number of parameters is incorrect, the malware will jump to the termination routine. If the argument count is correct, the malware proceeds to execute the sub_401070 function

The 0x401070 was previously seen when malware created XPS subkey using -in parameter.

This time, when we pass 4 parameters, it overwrites the Configuration current value with four new passed parameters. This way the malware can change its settings from the C2 commands.

2.3.4 -cc parameter

To execute this parameter, we need to also add the password. This function requires total of three passed parameters.

Lab09-01 -cc abcd

Opens registry XPS/Parameter, queries data, saves it.

Prints to the console a string “k:ups h:http://www.practicalmalwareanalysis.com p:80 per:60” which might sign current config.

Note: if we previously ran the -c parameter, we would see: “k:par1 h:par2 p:par3 per:par4”.

2.4. Backdoor analysis

Continuing analysis of path sub_402360 which is accessible by providing only one parameter.

First off it calls sub_401280 which picks up latest configuration from XPS/Configuration registry, then after successful access, it moves to sub_402020.

Sub_402020 has a lot of code in it, there are different code blocks with decrypted by IDA strings like:

- SLEEP – suspends current thread for one minute.
- UPLOAD – opens a file, reads the data from C2, overwrites a file with received data.
- DOWNLOAD – opens a file, reads the bytes, passes the bytes into buffer and sends it to C2.
- CMD – executes a command passed by the author via C2.
- NOTHING – just returns from the function

These functionalities collectively enable the backdoor to perform remote operations, manage files, execute commands, and maintain stealth through periodic sleeps, illustrating its capabilities as a versatile tool for unauthorized access and control.

3. Questions & Answers

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

To install the malware we need to pass the “-in” parameter along with the password “acbd”

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

-in

-re

-cc

Password has to be 4 characters and must be equal to “abcd”.

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

Change instruction at 0x402B38 JNZ short loc_402B3F to JMP.

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

- Existence of service called as the original file with appended “Service Manager”.
- Existence of the file at system32 directory.
- Existence of subkey “XPS” at HKLM\SOFTWARE\WOW6432Node\Microsoft\.

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

- SLEEP – suspends current thread for one minute.
- UPLOAD – opens a file, reads the data from C2, overwrites a file with received data.
- DOWNLOAD – opens a file, reads the bytes, passes the bytes into buffer and sends it to C2.
- CMD – executes a command passed by the author via C2.
- NOTHING – just returns from the function

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

Hostname: https://www.practicalmalwareanalysis.com

GET requests: aaaa/aaaa.aaa.