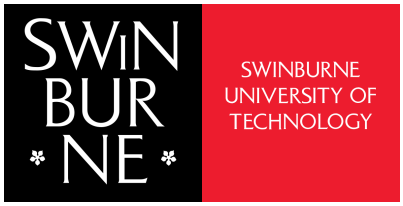


Swinburne University of Technology

COS20030 Malware Analysis

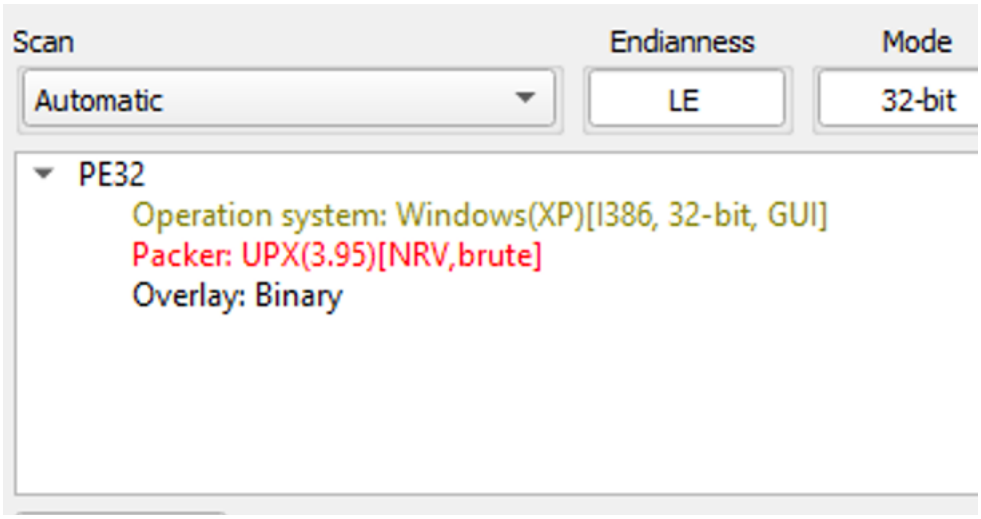
Assignment 1

Marco Giacoppo (104071453)

Section 1: Q1.exe

1) UPX was used.

- size difference between size and virtual size on UPX0
- few imported DLL files
- high entropy on PE Header and UPX1.



2) A) ebf55c5acad3066eb25a291faf3b4086

Name	Offset	Size	Hash
PE Header	00000000	00001000	ebf55c5acad3066eb25a291faf3b4086

B) 7.39257 (inside entropy in DIE)

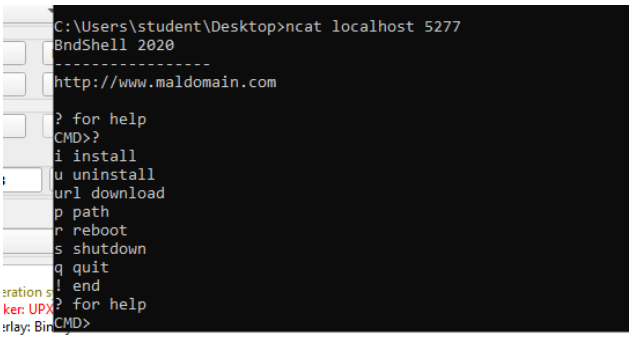
Regions

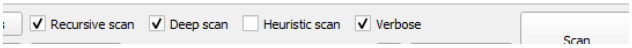
Offset	Size	Entropy	Status	Name
00000000	00001000	7.39257	packed	PE Header
00000200	00001200	7.51942	packed	Section(1)['UPX1']

C) 2001 – 11 -21 04:17:53

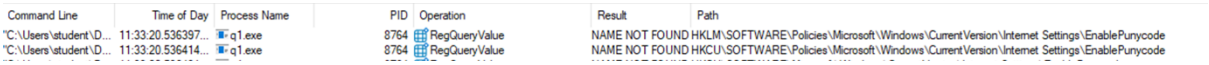
3) The `Q1.exe` malware is listening on port 5277. I identified this by running the `netstat -an | find “LISTEN” ` command in the command prompt, which displayed all the open ports on the system. Among the listed ports, 5277 was identified as the one `Q1.exe` was using for listening to incoming connections. I confirmed this by matching the PID associated with this port to the PID of `Q1.exe` using Process Explorer.

4) A)





B) "In Process Monitor, I observed that Q1.exe accessed the registry path **HKLM\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Internet Settings\EnablePunyCode**



- It's scanning through web applications probably from Microsoft Edge.

5) During the analysis, Q1.exe attempted to create a persistence mechanism by writing to the registry key **HKLM\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\bndshell**. This path was observed in the Detail column of Process Monitor when the registry key was being created.

6) Unpacking Q1 File

7) The malware uses the URLDownloadToFile API to download files from the specified URL

Section 2: Q2.exe

1) 4 files (KERNEL32.dll, ntdll.dll, RPCRT4.dll, and SHELL32.dll)

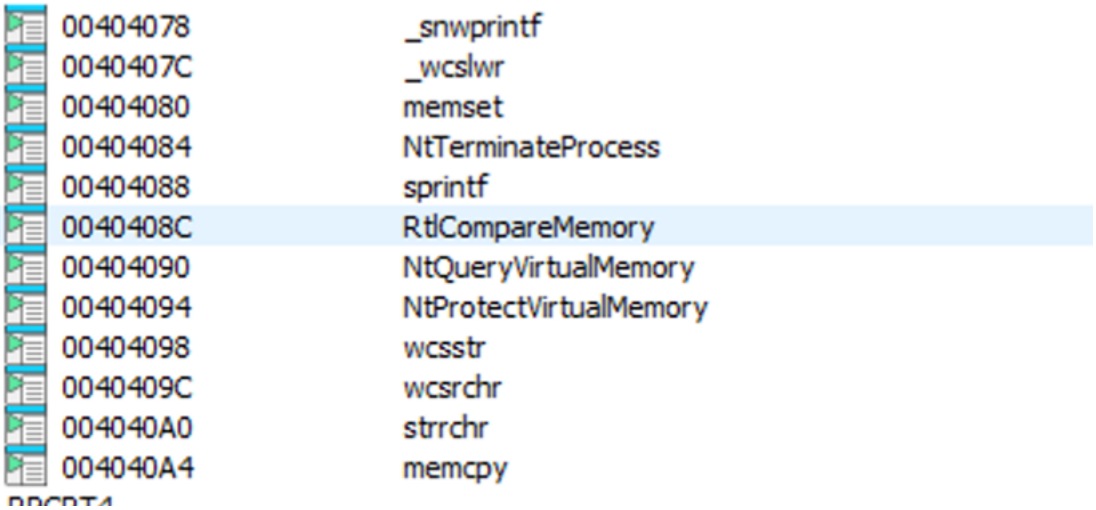
#	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	Hash	Name
0	00004548	00000000	00000000	00004780	00004000	886d8781	KERNEL32.dll
1	000045c0	00000000	00000000	00004826	00004078	fc2d9eb4	ntdll.dll
2	000045ac	00000000	00000000	0000486a	00004064	04ef9858	RPCRT4.dll
3	000045b8	00000000	00000000	00004886	00004070	abde73fb	SHELL32.dll

2) The address is '00404448'

Number	Offset	Address		Size	Type	String
37	00003848	00404448	Section(1)['.rdata']	07	A	ComSpec

3) The function FUN_00401d30 takes 1 argument which is "ntdll.dll"

4) NtTerminateProcess, NtQueryVirutalMemory, and NtProtectVirtualMemory are not documented.

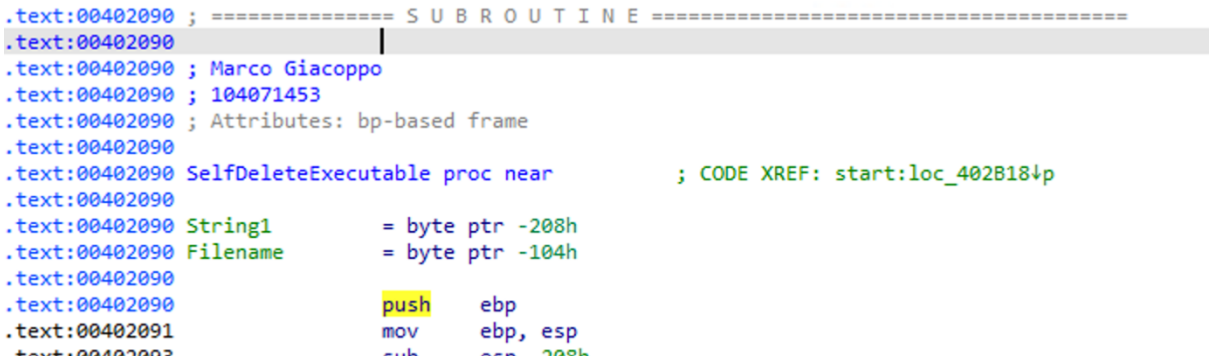


5) The parameter `param_1` was renamed to `LoadModuleIfNotLoaded` to reflect its purpose in the function. The function uses `param_1` to check if a specific module is already loaded into memory via `GetModuleHandleA`. If the module is not loaded, the function then uses `param_1` to load the module via `LoadLibraryA`.

```
.text:00401D30 ; This function checks if the module is loaded. If not, it loads the module using LoadLibraryA.
.text:00401D30 ;
.text:00401D30 ; Attributes: bp-based frame
.text:00401D30
.text:00401D30 ; int __cdecl sub_401D30(LPCSTR LoadModuleIfNotLoaded)
.text:00401D30 sub_401D30      proc near               ; CODE XREF: start+24↓p
.text:00401D30
.text:00401D30 LoadModuleIfNotLoaded= dword ptr  8
.text:00401D30
.text:00401D30                 push    ebp
.text:00401D31                 mov     ebp, esp
.text:00401D33                 push    esi
.text:00401D34                 mov     esi, [ebp+LoadModuleIfNotLoaded]
.text:00401D37                 push    esi             ; lpModuleName
.text:00401D38                 call    ds:GetModuleHandleA
.text:00401D3E                 test    eax, eax
```

6) The code between 0x0040239d and 0x004023b9 performs several checks and operations to determine whether the current process is running under WoW64 (a 32-bit process on a 64-bit system).
7) The function at 0x402090 is responsible for preparing a command-line string that can be used to delete the current executable file. Given these operations, the

function has been renamed to SelfDeleteExecutable.



Reflection on the Assignment:

This assignment was quite tricky, it forces us to be very careful in reverse engineering and binary analysis using few tools. One of the main challenges was interpreting the assembly code and mapping it to higher-level logic without any prior context or variable names. It required a careful examination of API calls to deduce the purpose of the function and its parameters. Finding the param_1 was also quite challenging since the IDA renamed it automatically to ipmodulename, which I found out after cross checking with Ghidra.

A key highlight was analyzing system-level interactions, such as identyfing how functions like IsWOw64Process and ShellExecuteA were used. The process of renaming and commenting on the self-deletion function provided insight into how programs or malware might clean up after execution. Overall, I'm pretty happy with how everything turned out.