Anti-analysis technique for PE Analysis Tools –INT Spoofing–

When analysing Windows executable file type (PE file) malware, a tool to parse and display the PE file's structure (hereafter "PE analysis tool") is often used. This tool enables referring to a list of APIs that the malware imports (Import API) and functions that it exports. By analysing the data, it is possible to presume the malware's function as in communicating with external servers or creating registry entries, etc. In this way, PE analysis tools are often used for malware analysis, however, a type of malware which has techniques to disturb operations of PE analysis tools has already been observed [1].

This entry introduces techniques to deceive analysts by displaying incorrect information in the Import API, and measures to implement in PE analysis tools against the issue.

INT (Import Name Table) and IAT (Import Address Table)

PE files contain 2 address tables related to Import API – INT and IAT. INT describes the address of the area which stores API names imported by the PE file. IAT is used when actually calling an API, and writes an entry address of the functions corresponding to the API when the module which exports the function is loaded. For more information about PE file formats, please refer to Microsoft's website [2].

NT header in a PE file describes various kinds of information required for executing the file. NT header is structured as "IMAGE_NT_HEADERS", and INT and IAT can be identified by tracing the address in "IMAGE_DATA_DIRECTORY" of Optional Header within the structure (Figure 1) [3].

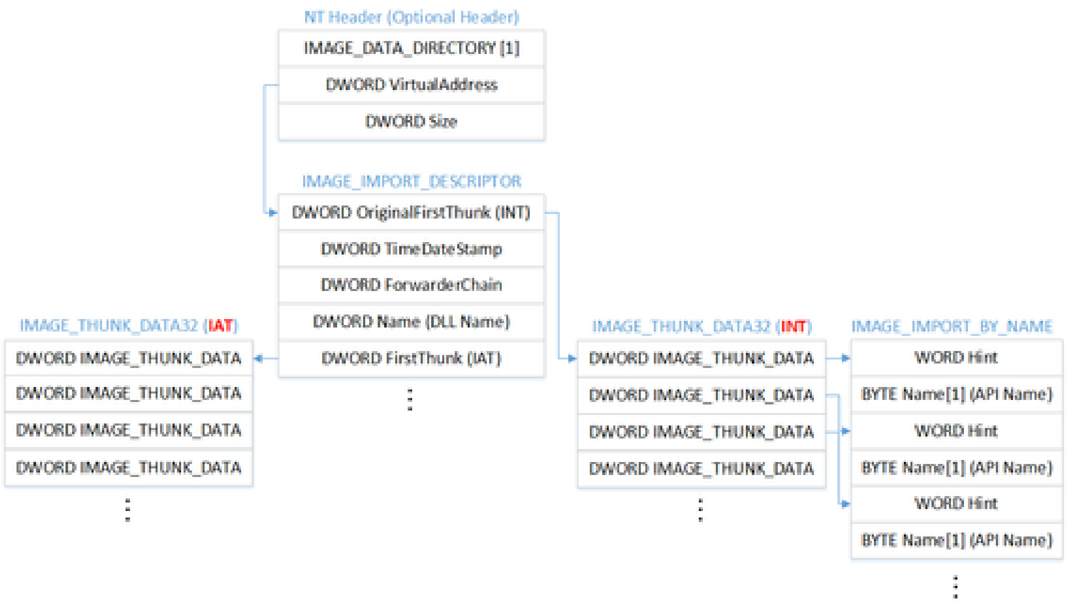
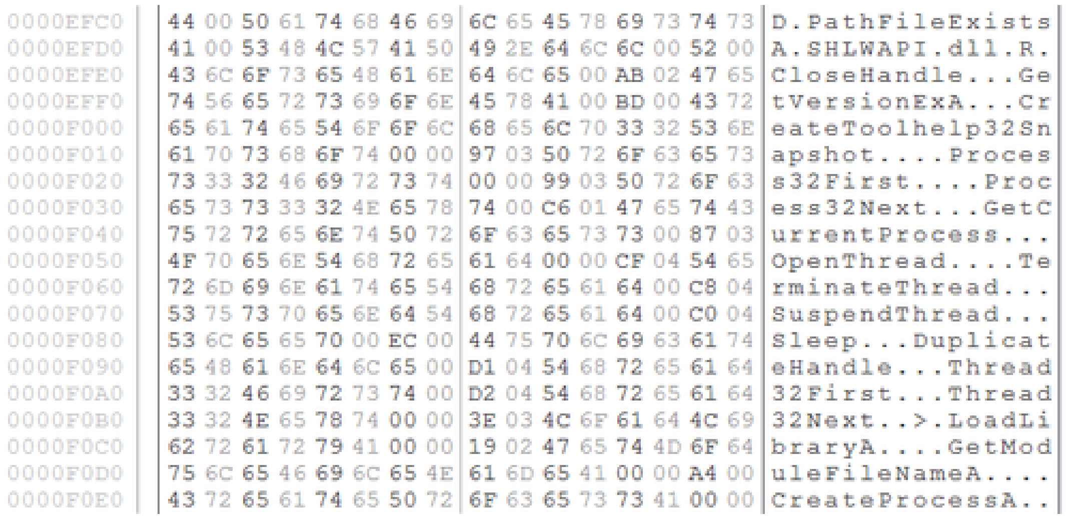


Figure 1: INT and IAT related section within NT header in a PE file

The Name field of "IMAGE_IMPORT_BY_NAME" structure, which is referred to by INT, describes importing API names as a string. Generally, IMAGE_IMPORT_BY_NAME lists API names in a sequence as in Figure 2.



```
0000EFC0 | 44 00 50 61 74 68 46 69 | 6C 65 45 78 69 73 74 73 | D.PathFileExists
0000EFD0 | 41 00 53 48 4C 57 41 50 | 49 2E 64 6C 6C 00 52 00 | A.SHLWAPI.dll.R.
0000EFE0 | 43 6C 6F 73 65 48 61 6E | 64 6C 65 00 AB 02 47 65 | CloseHandle...Ge
0000EFF0 | 74 56 65 72 73 69 6F 6E | 45 78 41 00 BD 00 43 72 | tVersionExA...Cr
0000F000 | 65 61 74 65 54 6F 6F 6C | 68 65 6C 70 33 32 53 6E | eateToolhelp32Sn
0000F010 | 61 70 73 68 6F 74 00 00 | 97 03 50 72 6F 63 65 73 | apshot....Proces
0000F020 | 73 33 32 46 69 72 73 74 | 00 00 99 03 50 72 6F 63 | s32First....Proc
0000F030 | 65 73 73 33 32 4E 65 78 | 74 00 C6 01 47 65 74 43 | ess32Next...GetC
0000F040 | 75 72 72 65 6E 74 50 72 | 6F 63 65 73 73 00 87 03 | urrentProcess...
0000F050 | 4F 70 65 6E 54 68 72 65 | 61 64 00 00 CF 04 54 65 | OpenThread....Te
0000F060 | 72 6D 69 6E 61 74 65 54 | 68 72 65 61 64 00 C8 04 | rminateThread...
0000F070 | 53 75 73 70 65 6E 64 54 | 68 72 65 61 64 00 C0 04 | SuspendThread...
0000F080 | 53 6C 65 65 70 00 EC 00 | 44 75 70 6C 69 63 61 74 | Sleep...Duplicat
0000F090 | 65 48 61 6E 64 6C 65 00 | D1 04 54 68 72 65 61 64 | eHandle...Thread
0000F0A0 | 33 32 46 69 72 73 74 00 | D2 04 54 68 72 65 61 64 | 32First...Thread
0000F0B0 | 33 32 4E 65 78 74 00 00 | 3E 03 4C 6F 61 64 4C 69 | 32Next..>.LoadLi
0000F0C0 | 62 72 61 72 79 41 00 00 | 19 02 47 65 74 4D 6F 64 | braryA....GetMod
0000F0D0 | 75 6C 65 46 69 6C 65 4E | 61 6D 65 41 00 00 A4 00 | uleFileNameA....
0000F0E0 | 43 72 65 61 74 65 50 72 | 6F 63 65 73 73 41 00 00 | CreateProcessA..
```

Figure 2: Example of IMAGE_IMPORT_BY_NAME

INT Spoofing

IMAGE_IMPORT_BY_NAME contains strings specifying API names. Even if someone tries to alter the API name in IMAGE_IMPORT_BY_NAME to disguise it as another PE file, it would not be executed properly since it would import unintended API when executing the PE file. As the red part in Figure 3 indicates, however, if the PE file is modified by adding new API names at the end of the INT to existing API names within the INT, it will not attempt to load a module since the IAT does not have a field that stores the entry address of the functions corresponding to the added API name. If PE analysis tools display such deliberately added API names, analysts would believe that the PE file has new APIs that is imported, which would confuse the analysis.

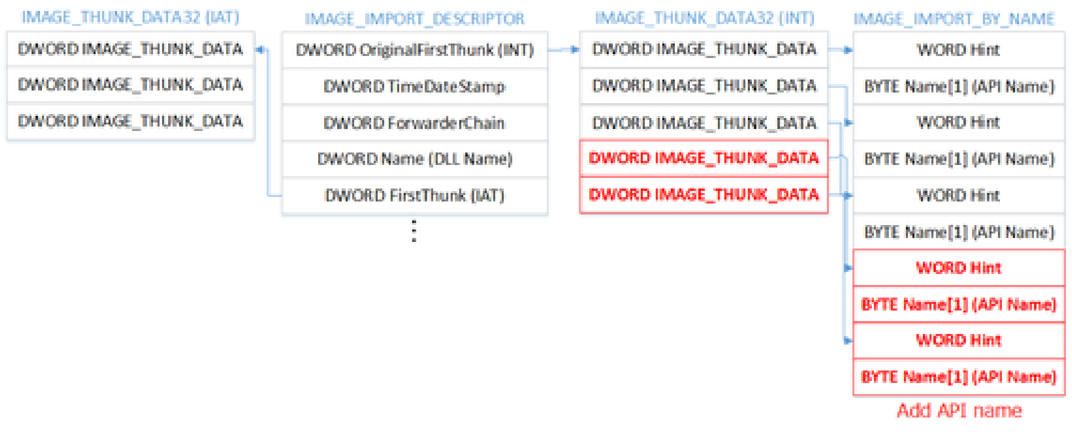
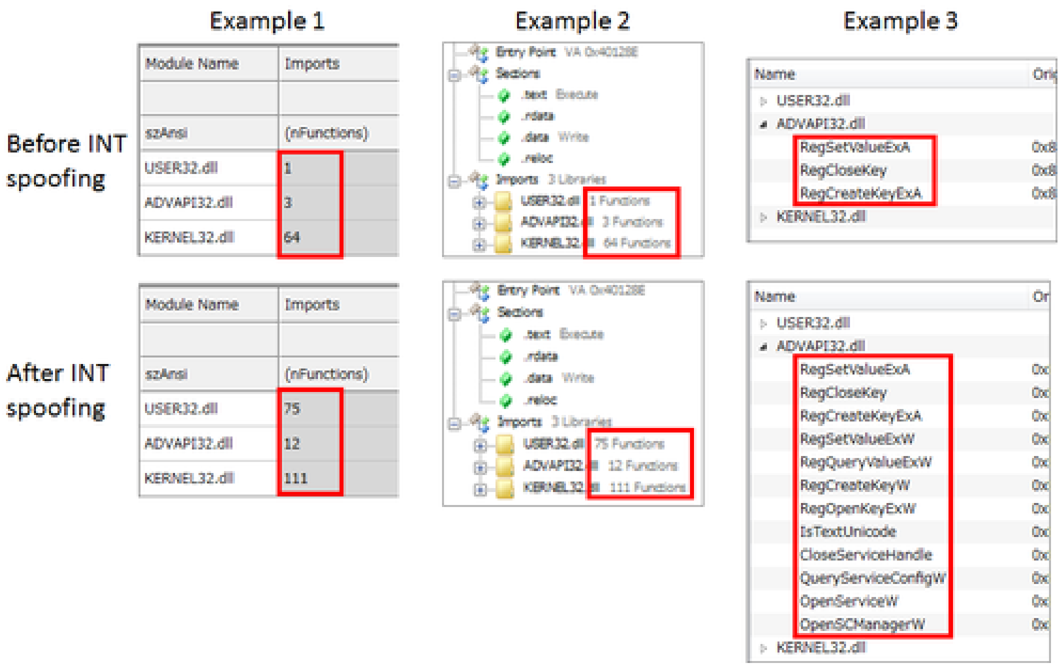


Figure 3: Example of INT spoofing

Check for INT-spoofed PE files using PE analysis tools

Many of the existing PE analysis tools refer to only INT when listing Import API, and recognise and display strings in IMAGE_IMPORT_BY_NAME as API names. When handling normal PE files, there is no issues with the behaviour since importing API addresses corresponding to the strings in IMAGE_IMPORT_BY_NAME, are written in the IAT.

However, if INT is spoofed by the above mentioned method, extra APIs are also listed. As an experiment, JPCERT/CC generated some INT-spoofed PE files, and tested how their Import API would be displayed in several PE analysis tools. As a result, many of them displayed extra APIs that are not actually imported.



Example 1, before INT spoofing:

Module Name	Imports
szAnsi	(nFunctions)
USER32.dll	1
ADVAPI32.dll	3
KERNEL32.dll	64

Example 1, after INT spoofing:

Module Name	Imports
szAnsi	(nFunctions)
USER32.dll	75
ADVAPI32.dll	12
KERNEL32.dll	111

```
Entry Point  VA 0x40128E
Sections
    .text  Execute
    .rdata
    .data  Write
    .reloc
Imports  3 Libraries
    USER32.dll  1 Functions
    ADVAPI32.dll  3 Functions
    KERNEL32.dll  64 Functions
```

```
Entry Point  VA 0x40128E
Sections
    .text  Execute
    .rdata
    .data  Write
    .reloc
Imports  3 Libraries
    USER32.dll  75 Functions
    ADVAPI32.dll  12 Functions
    KERNEL32.dll  111 Functions
```

```
Name
  USER32.dll
  ADVAPI32.dll
      RegSetValueExA
      RegCloseKey
      RegCreateKeyExA
  KERNEL32.dll
```

```
Name
  USER32.dll
  ADVAPI32.dll
      RegSetValueExA
      RegCloseKey
      RegCreateKeyExA
      RegSetValueExW
      RegQueryValueExW
      RegCreateKeyW
      RegOpenKeyExW
      IsTextUnicode
      CloseServiceHandle
      QueryServiceConfigW
      OpenServiceW
      OpenSCManagerW
  KERNEL32.dll
```

Figure 4: Analysis examples of INT-spoofed executable files on PE analysis tools
(Indicates the number of Import API increased due to INT spoofing)

Countermeasures against INT spoofing

One countermeasure against such spoofing would be to compare INT and IAT on a PE analysis tool and only display APIs that are actually imported (and not display added API names marked in red in the Figure 3). pyimpfuzzy, which was introduced in a past blog entry, is also a tool which performs analysis based on Import API. In its first version, there was an issue where INT-spoofed samples could not be analysed correctly. As such, the tool was updated with a new feature to compare INT and IAT, and only analyse the APIs that are actually imported.

Many PE analysis tools display strings in IMAGE_IMPORT_BY_NAME as they are. However, many debuggers and IDA refer to IATs when displaying Import API, and thus most of them do not seem to be affected by INT spoofing. When referring to the information on Import API in malware analysis, it is recommended to check APIs that are actually loaded in IAT by using a debugger, as well as INT strings.

Summary

JPCERT/CC has not yet observed any INT-spoofed samples, however, this disguising technique could possibly be abused in the near future. Automated analysis tools based on Import API may be affected by INT spoofing. As introduced above, pyimpfuzzy has been updated to a new version – please make sure that you are using the latest version (version 0.02).

Thanks for reading.

- Shusei Tomonaga
(Translated by Yukako Uchida)

References:

[1] Palo Alto Networks - The Dukes R&D Finds a New Anti-Analysis Technique

http://researchcenter.paloaltonetworks.com/2016/09/unit42-the-dukes-rd-finds-a-new-anti-analysis-technique/
[2] Microsoft - PE Format
https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx?f=255&MSPPError=-2147217396
[3] Microsoft - IMAGE_NT_HEADERS structure
https://msdn.microsoft.com/en-us/library/windows/desktop/ms680336(v=vs.85).aspx