09/11/2025

Malicious Code Analysis

Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Overview

01 Data Directories

02 Examples

Part One

01

Data Directories

2025-8-23

Data Directories

This section contains information about the various data directories used by the operating system, including import and export tables, resources, and relocations.

Data Directories

- Data Directory is a piece of data located within one of the sections of the PE file.
- IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES].
- IMAGE_NUMBEROF_DIRECTORY_ENTRIES is a constant defined with the value 16, meaning that this array can have up to 16 IMAGE_DATA_DIRECTORY entries:

```
#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16
```

Data Directories

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

Data Directories

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT        0   // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT        1   // Import Directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE      2   // Resource Directory
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION     3   // Exception Directory
#define IMAGE_DIRECTORY_ENTRY_SECURITY      4   // Security Directory
#define IMAGE_DIRECTORY_ENTRY_BASERELOC     5   // Base Relocation Table
#define IMAGE_DIRECTORY_ENTRY_DEBUG         6   // Debug Directory
//      IMAGE_DIRECTORY_ENTRY_COPYRIGHT     7   // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE  7   // Architecture Specific Data
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR     8   // RVA of GP
#define IMAGE_DIRECTORY_ENTRY_TLS           9   // TLS Directory
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG   10  // Load Configuration Directory
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT  11  // Bound Import Directory in headers
#define IMAGE_DIRECTORY_ENTRY_IAT           12  // Import Address Table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT  13  // Delay Load Import Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14 // COM Runtime descriptor
```

Data Directories

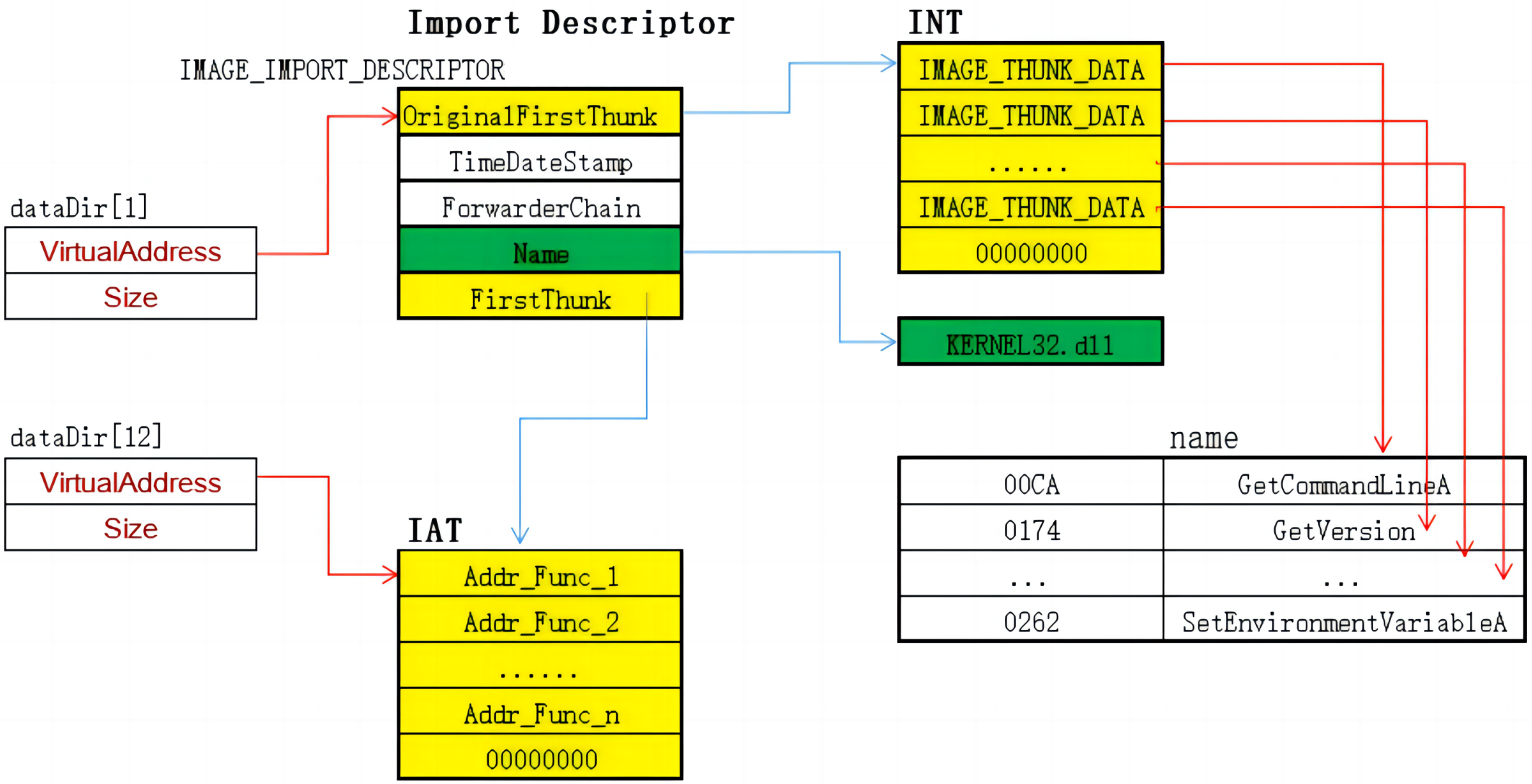
Disasm: .rdata | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hd

Offset	Name	Value	Value
180	Loader Flags	0	
184	Number of RVAs and Sizes	10	
	Data Directory	Address	Size
188	Export Directory	0	0
190	Import Directory	27AC	B4
198	Resource Directory	5000	1E0
1A0	Exception Directory	4000	168
1A8	Security Directory	0	0
1B0	Base Relocation Table	6000	28
1B8	Debug Directory	2248	70
1C0	Architecture Specific Data	0	0
1C8	RVA of GlobalPtr	0	0
1D0	TLS Directory	0	0
1D8	Load Configuration Directory	22C0	130
1E0	Bound Import Directory in headers	0	0
1E8	Import Address Table	2000	198
1F0	Delay Load Import Descriptors	0	0
1F8	.NET header	0	0

Import Directory Table

- The Import Directory Table is a Data Directory located at the .idata section.
- It consists of an array of IMAGE_IMPORT_DESCRIPTOR structures, each one of them is for a DLL.
- It doesn’t have a fixed size, so the last IMAGE_IMPORT_DESCRIPTOR of the array is zeroed-out (NULL-Padded) to indicate the end of the Import Directory Table.

Summary


Import Descriptor
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
dataDir[1]
VirtualAddress
Size
dataDir[12]
VirtualAddress
Size
IAT
Addr_Func_1
Addr_Func_2
.....
Addr_Func_n
00000000
INT
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
.....
IMAGE_THUNK_DATA
00000000
KERNEL32.dll
name
00CA
GetCommandLineA
0174
GetVersion
...
...
0262
SetEnvironmentVariableA

Import Directory Table

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;
        DWORD   OriginalFirstThunk;
    } DUMMYUNIONNAME;
    DWORD   TimeDateStamp;
    DWORD   ForwarderChain;
    DWORD   Name;
    DWORD   FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```

Fields

- **OriginalFirstThunk:** RVA of the ILT.
- **TimeDateStamp:** A time date stamp, that's initially set to 0 if not bound and set to -1 if bound.
  - In case of an unbound import the time date stamp gets updated to the time date stamp of the DLL after the image is bound.
  - In case of a bound import it stays set to -1 and the real time date stamp of the DLL can be found in the Bound Import Directory Table in the corresponding IMAGE_BOUND_IMPORT_DESCRIPTOR.
- **ForwarderChain:** The index of the first forwarder chain reference. This is something responsible for DLL forwarding. *(DLL forwarding is when a DLL forwards some of its exported functions to another DLL.)*
- **Name:** An RVA of an ASCII string that contains the name of the imported DLL.
- **FirstThunk:** RVA of the IAT.

Bound Imports

- Bound import essentially means that the import table contains fixed addresses for the imported functions. These addresses are calculated and written during compile time by the linker.
- Using bound imports is a speed optimization, it reduces the time needed by the loader to resolve function addresses and fill the IAT, however if at run-time the bound addresses do not match the real ones then the loader will have to resolve these addresses again and fix the IAT.
- When discussing IMAGE_IMPORT_DESCRIPTOR.TimeDateStamp, I mentioned that in case of a bound import, the time date stamp is set to -1 and the real time date stamp of the DLL can be found in the corresponding IMAGE_BOUND_IMPORT_DESCRIPTOR in the Bound Import Data Directory.

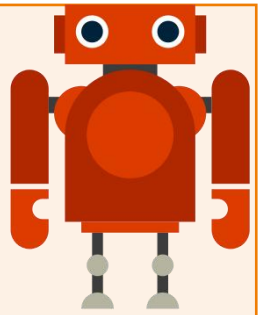
Bound Import Data Directory

- The Bound Import Data Directory is similar to the Import Directory Table, however as the name suggests, it holds information about the bound imports.

- It consists of an array of IMAGE_BOUND_IMPORT_DESCRIPTOR structures, and ends with a zeroed-out IMAGE_BOUND_IMPORT_DESCRIPTOR.

Bound Import Data Directory

```
typedef struct _IMAGE_BOUND_IMPORT_DESCRIPTOR {
    DWORD   TimeDateStamp;
    WORD    OffsetModuleName;
    WORD    NumberOfModuleForwarderRefs;
// Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
} IMAGE_BOUND_IMPORT_DESCRIPTOR,  *PIMAGE_BOUND_IMPORT_DESCRIPTOR;
```



Fields

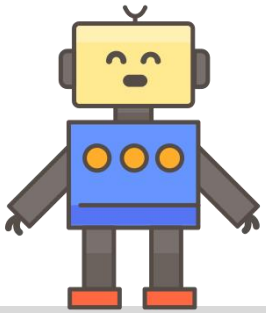
- **TimeDateStamp:** The time date stamp of the imported DLL.
- **OffsetModuleName:** An offset to a string with the name of the imported DLL. It's an offset from the first IMAGE_BOUND_IMPORT_DESCRIPTOR.
- **NumberOfModuleForwarderRefs:** The number of the IMAGE_BOUND_FORWARDER_REF structures that immediately follow this structure.
- IMAGE_BOUND_FORWARDER_REF is a structure that's identical to IMAGE_BOUND_IMPORT_DESCRIPTOR, the only difference is that the last member is reserved.

Import Lookup Table (ILT)

- Sometimes people refer to it as the Import Name Table (INT).
- Every imported DLL has an Import Lookup Table.
- IMAGE_IMPORT_DESCRIPTOR.OriginalFirstThunk holds the RVA of the ILT of the corresponding DLL.
- The ILT is essentially a table of names or references, it tells the loader which functions are needed from the imported DLL.
- The ILT consists of an array of 32-bit numbers (for PE32) or 64-bit numbers for (PE32+), the last one is zeroed-out to indicate the end of the ILT.

Import Lookup Table (ILT)

- Each entry of these entries encodes information as follows:
  - **Bit 31/63 (most significant bit):** This is called the Ordinal/Name flag, it specifies whether to import the function by name or by ordinal.
  - **Bits 15-0:** If the Ordinal/Name flag is set to 1 these bits are used to hold the 16-bit ordinal number that will be used to import the function, bits 30-15/62-15 for PE32/PE32+ must be set to 0.
  - **Bits 30-0:** If the Ordinal/Name flag is set to 0 these bits are used to hold an RVA of a Hint/Name table.



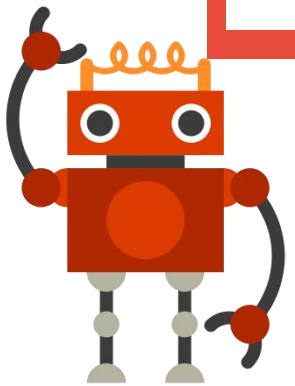
Hint/Name Table

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    CHAR   Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

- **Hint:** A word that contains a number, this number is used to look-up the function, that number is first used as an index into the export name pointer table, if that initial check fails a binary search is performed on the DLL's export name pointer table.
- **Name:** A null-terminated string that contains the name of the function to import.

Import Address Table (IAT)

- On disk, the IAT is identical to the ILT, however during bounding when the binary is being loaded into memory, the entries of the IAT get overwritten with the addresses of the functions that are being imported.



Summary

- So to summarize what we discussed in this post, for every DLL the executable is loading functions from, there will be an IMAGE_IMPORT_DESCRIPTOR within the Image Directory Table. The IMAGE_IMPORT_DESCRIPTOR will contain the name of the DLL, and two fields holding RVAs of the ILT and the IAT.
- The ILT will contain references for all the functions that are being imported from the DLL. The IAT will be identical to the ILT until the executable is loaded in memory, then the loader will fill the IAT with the actual addresses of the imported functions.
- If the DLL import is a bound import, then the import information will be contained in IMAGE_BOUND_IMPORT_DESCRIPTOR structures in a separate Data Directory called the Bound Import Data Directory.

Data Directories

Disasm: .rdata | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hd

Offset	Name	Value	Value
180	Loader Flags	0	
184	Number of RVAs and Sizes	10	
	Data Directory	Address	Size
188	Export Directory	0	0
190	Import Directory	27AC	B4
198	Resource Directory	5000	1E0
1A0	Exception Directory	4000	168
1A8	Security Directory	0	0
1B0	Base Relocation Table	6000	28
1B8	Debug Directory	2248	70
1C0	Architecture Specific Data	0	0
1C8	RVA of GlobalPtr	0	0
1D0	TLS Directory	0	0
1D8	Load Configuration Directory	22C0	130
1E0	Bound Import Directory in headers	0	0
1E8	Import Address Table	2000	198
1F0	Delay Load Import Descriptors	0	0
1F8	.NET header	0	0

Section Table

Disasm: .text | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | Imports | Resources | Ex

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	400	E00	1000	D2C	60000020	0	0	0
>	1200	^	1D2C	^	r-x			
.rdata	1200	1000	2000	E3C	40000040	0	0	0
>	2200	^	2E3C	^	r--			
.data	2200	200	3000	638	C0000040	0	0	0
>	2400	^	3638	^	rw-			
.pdata	2400	200	4000	168	40000040	0	0	0
>	2600	^	4168	^	r--			
.rsrc	2600	200	5000	1E0	40000040	0	0	0
>	2800	^	51E0	^	r--			
.reloc	2800	200	6000	28	42000040	0	0	0
>	2A00	^	6028	^	r--			

Summary

Disasm: .rdata | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | Imports | Resources | Exception | Base

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
19AC	USER32.dll	1	FALSE	28E0	0	0	2A06	2080
19C0	VCRUNTI...	4	FALSE	28F0	0	0	2A68	2090
19D4	api-ms-wi...	18	FALSE	2948	0	0	2C2C	20E8
19E8	api-ms-wi...	1	FALSE	2938	0	0	2C4E	20D8
19FC	api-ms-wi...	2	FALSE	29E0	0	0	2C6E	2180
1A10	api-ms-wi...	1	FALSE	2928	0	0	2C8E	20C8
1A24	api-ms-wi...	1	FALSE	2918	0	0	2CB0	20B8
1A38	KERNEL32....	15	FALSE	2860	0	0	2E2E	2000

Part Two

02

Example

2025-8-23

Section Table

Disasm: .text | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | Imports | Resources | Ex

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	400	E00	1000	D2C	60000020	0	0	0
>	1200	^	1D2C	^	r-x			
.rdata	1200	1000	2000	E3C	40000040	0	0	0
>	2200	^	2E3C	^	r--			
.data	2200	200	3000	638	C0000040	0	0	0
>	2400	^	3638	^	rw-			
.pdata	2400	200	4000	168	40000040	0	0	0
>	2600	^	4168	^	r--			
.rsrc	2600	200	5000	1E0	40000040	0	0	0
>	2800	^	51E0	^	r--			
.reloc	2800	200	6000	28	42000040	0	0	0
>	2A00	^	6028	^	r--			

Example

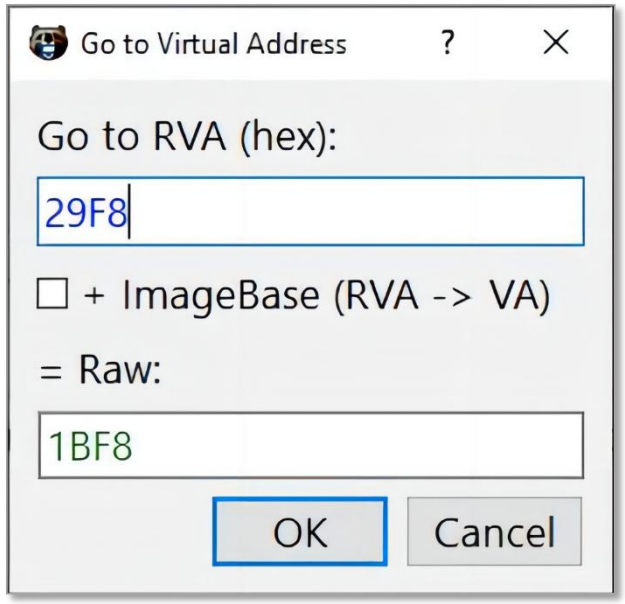
- For example, if we take USER32.dll and follow the RVA of its ILT (referenced by OriginalFirstThunk), we'll find only 1 entry (because only one function is imported), and that entry looks like this:

	0	1	2	3	4	5	6	7	8
1AE0	F8	29	00	00	00	00	00	00	00

- This is a 64-bit executable, so the entry is 64 bits long.
- As you can see, the last byte is set to 0, indicating that a Hint/Table name should be used to look-up the function.

Examples

- We know that the RVA of this Hint/Table name should be referenced by the first 2 bytes, so we should follow RVA 0x29F8:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1BF8	83	02	4D	65	73	73	61	67	65	42	6F	78	41	00	55	53	.	.	M	e	s	s	a	g	e	B	o	x	A	.	U	S
1C08	45	52	33	32	2E	64	6C	6C	00	00	08	00	5F	5F	43	5F	E	R	3	2	.	d	l	l	.	.	.	.	_	_	C	_

Example

- Now we’re looking at an IMAGE_IMPORT_BY_NAME structure, first two bytes hold the hint, which is 0x283, the rest of the structure holds the full name of the function which is MessageBoxA.
- We can verify that our interpretation of the data is correct by looking at how PE-bear parsed it, and we’ll see the same results:

USER32.dll [1 entry]

Call via	Name	Ordinal	Original Thu	Thunk	Forwarder	Hint
2080	MessageBoxA	-	29F8	29F8	-	283

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

09/11/2025