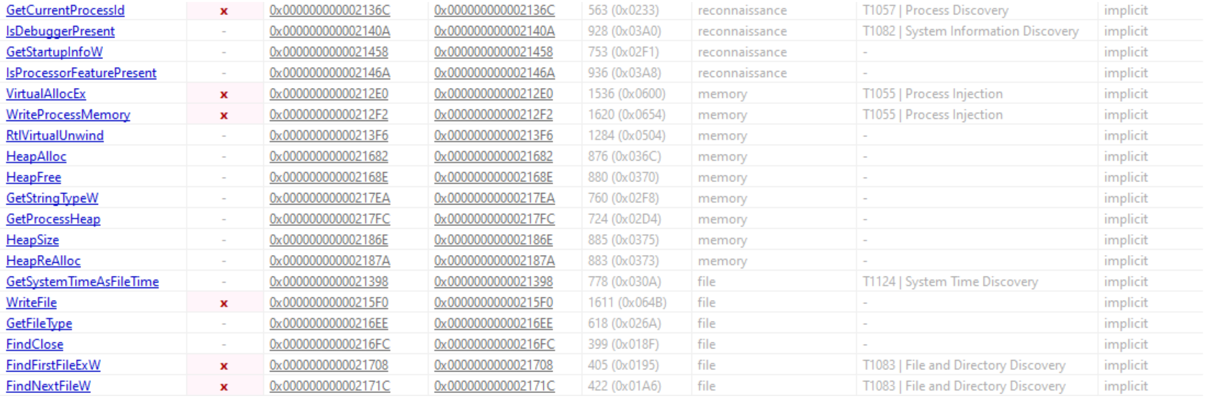
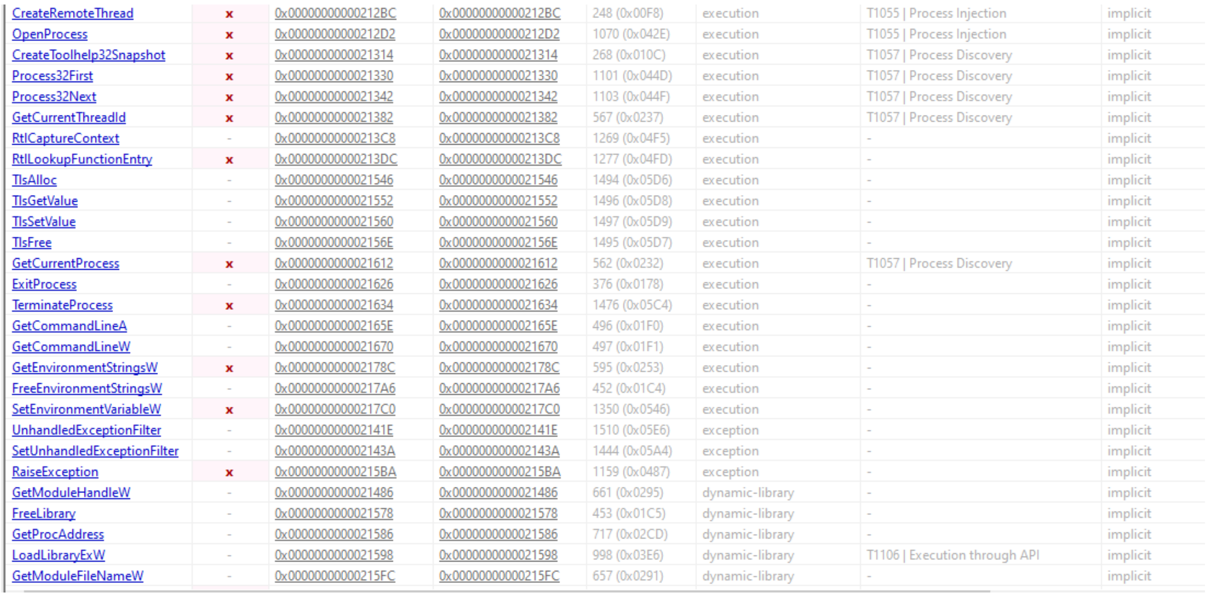
Reverse Engineering Process Injection

To verify whether the program is a Process Injection or not, we load the file in the pestudio, and then check the imported functions, for process injection, we need functions like VirtualAllocEx, WriteProcessMemory, CreateProcessthread, OpenProcess, etc.,

In pestudio we can see that:



GetCurrentProcessId	x	0x000000000002136C	0x000000000002136C	563 (0x0233)	reconnaissance	T1057 | Process Discovery	implicit
IsDebuggerPresent	-	0x000000000002140A	0x000000000002140A	928 (0x03A0)	reconnaissance	T1082 | System Information Discovery	implicit
GetStartupInfoW	-	0x0000000000021458	0x0000000000021458	753 (0x02F1)	reconnaissance	-	implicit
IsProcessorFeaturePresent	-	0x000000000002146A	0x000000000002146A	936 (0x03A8)	reconnaissance	-	implicit
VirtualAllocEx	x	0x00000000000212E0	0x00000000000212E0	1536 (0x0600)	memory	T1055 | Process Injection	implicit
WriteProcessMemory	x	0x00000000000212F2	0x00000000000212F2	1620 (0x0654)	memory	T1055 | Process Injection	implicit
RtlVirtualUnwind	-	0x00000000000213F6	0x00000000000213F6	1284 (0x0504)	memory	-	implicit
HeapAlloc	-	0x0000000000021682	0x0000000000021682	876 (0x036C)	memory	-	implicit
HeapFree	-	0x000000000002168E	0x000000000002168E	880 (0x0370)	memory	-	implicit
GetStringTypeW	-	0x00000000000217EA	0x00000000000217EA	760 (0x02F8)	memory	-	implicit
GetProcessHeap	-	0x00000000000217FC	0x00000000000217FC	724 (0x02D4)	memory	-	implicit
HeapSize	-	0x000000000002186E	0x000000000002186E	885 (0x0375)	memory	-	implicit
HeapReAlloc	-	0x000000000002187A	0x000000000002187A	883 (0x0373)	memory	-	implicit
GetSystemTimeAsFileTime	-	0x0000000000021398	0x0000000000021398	778 (0x030A)	file	T1124 | System Time Discovery	implicit
WriteFile	x	0x00000000000215F0	0x00000000000215F0	1611 (0x064B)	file	-	implicit
GetFileType	-	0x00000000000216EE	0x00000000000216EE	618 (0x026A)	file	-	implicit
FindClose	-	0x00000000000216FC	0x00000000000216FC	399 (0x018F)	file	-	implicit
FindFirstFileExW	x	0x0000000000021708	0x0000000000021708	405 (0x0195)	file	T1083 | File and Directory Discovery	implicit
FindNextFileW	x	0x000000000002171C	0x000000000002171C	422 (0x01A6)	file	T1083 | File and Directory Discovery	implicit



CreateRemoteThread	x	0x00000000000212BC	0x00000000000212BC	248 (0x00F8)	execution	T1055 | Process Injection	implicit
OpenProcess	x	0x00000000000212D2	0x00000000000212D2	1070 (0x042E)	execution	T1055 | Process Injection	implicit
CreateToolhelp32Snapshot	x	0x0000000000021314	0x0000000000021314	268 (0x010C)	execution	T1057 | Process Discovery	implicit
Process32First	x	0x0000000000021330	0x0000000000021330	1101 (0x044D)	execution	T1057 | Process Discovery	implicit
Process32Next	x	0x0000000000021342	0x0000000000021342	1103 (0x044F)	execution	T1057 | Process Discovery	implicit
GetCurrentThreadId	x	0x0000000000021382	0x0000000000021382	567 (0x0237)	execution	T1057 | Process Discovery	implicit
RtlCaptureContext	-	0x00000000000213C8	0x00000000000213C8	1269 (0x04F5)	execution	-	implicit
RtlLookupFunctionEntry	x	0x00000000000213DC	0x00000000000213DC	1277 (0x04FD)	execution	-	implicit
TlsAlloc	-	0x0000000000021546	0x0000000000021546	1494 (0x05D6)	execution	-	implicit
TlsGetValue	-	0x0000000000021552	0x0000000000021552	1496 (0x05D8)	execution	-	implicit
TlsSetValue	-	0x0000000000021560	0x0000000000021560	1497 (0x05D9)	execution	-	implicit
TlsFree	-	0x000000000002156E	0x000000000002156E	1495 (0x05D7)	execution	-	implicit
GetCurrentProcess	x	0x0000000000021612	0x0000000000021612	562 (0x0232)	execution	T1057 | Process Discovery	implicit
ExitProcess	-	0x0000000000021626	0x0000000000021626	376 (0x0178)	execution	-	implicit
TerminateProcess	x	0x0000000000021634	0x0000000000021634	1476 (0x05C4)	execution	-	implicit
GetCommandLineA	-	0x000000000002165E	0x000000000002165E	496 (0x01F0)	execution	-	implicit
GetCommandLineW	-	0x0000000000021670	0x0000000000021670	497 (0x01F1)	execution	-	implicit
GetEnvironmentStringsW	x	0x000000000002178C	0x000000000002178C	595 (0x0253)	execution	-	implicit
FreeEnvironmentStringsW	-	0x00000000000217A6	0x00000000000217A6	452 (0x01C4)	execution	-	implicit
SetEnvironmentVariableW	x	0x00000000000217C0	0x00000000000217C0	1350 (0x0546)	execution	-	implicit
UnhandledExceptionFilter	-	0x000000000002141E	0x000000000002141E	1510 (0x05E6)	exception	-	implicit
SetUnhandledExceptionFilter	-	0x000000000002143A	0x000000000002143A	1444 (0x05A4)	exception	-	implicit
RaiseException	x	0x00000000000215BA	0x00000000000215BA	1159 (0x0487)	exception	-	implicit
GetModuleHandleW	-	0x0000000000021486	0x0000000000021486	661 (0x0295)	dynamic-library	-	implicit
FreeLibrary	-	0x0000000000021578	0x0000000000021578	453 (0x01C5)	dynamic-library	-	implicit
GetProcAddress	-	0x0000000000021586	0x0000000000021586	717 (0x02CD)	dynamic-library	-	implicit
LoadLibraryExW	-	0x0000000000021598	0x0000000000021598	998 (0x03E6)	dynamic-library	T1106 | Execution through API	implicit
GetModuleFileNameW	-	0x00000000000215FC	0x00000000000215FC	657 (0x0291)	dynamic-library	-	implicit

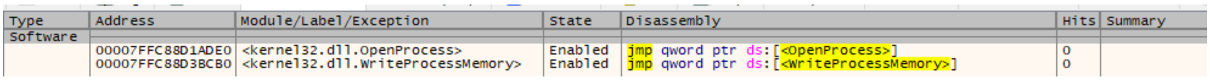
Lots of imported files, which has been flagged by pestudio.

So, we know that it is a process injection, so we can just put the breakpoints, and then analyze the program, by putting the breakpoints at VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, OpenProcess.

We don’t know what program this .exe file is looking for; we can still find it.

For simplicity let’s open the mspaint, and just leave it.

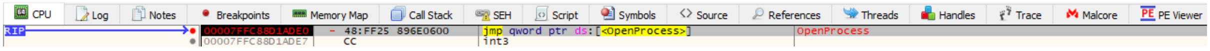
Now open the xdbg, and open the process injection program, and add the breakpoint at OpenProcess, and at WriteProcessMemory.



```
Type      Address           Module/Label/Exception              State    Disassembly                             Hits Summary
Software
          00007FFC88D1ADE0  <kernel32.dll.OpenProcess>          Enabled  jmp qword ptr ds:[<OpenProcess>]        0
          00007FFC88D3BCB0  <kernel32.dll.WriteProcessMemory>   Enabled  jmp qword ptr ds:[<WriteProcessMemory>] 0
```

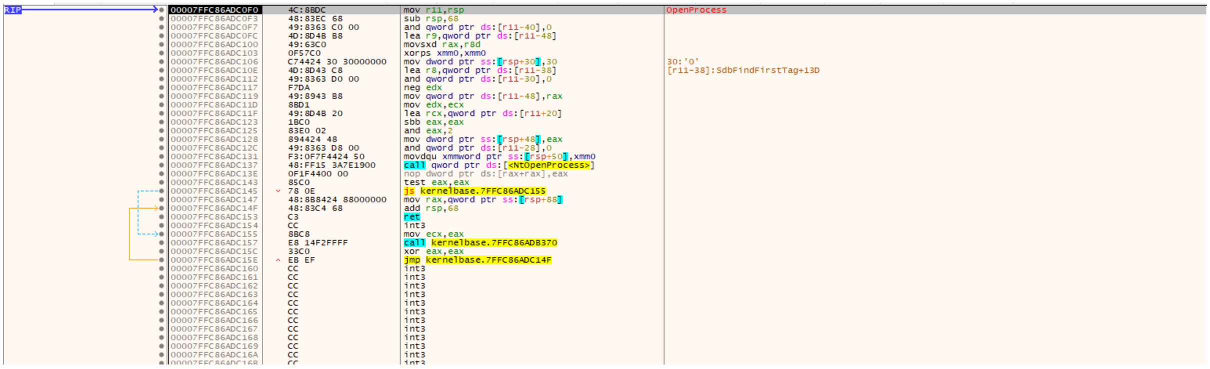
Now we have added the breakpoint, now tun the program.

And we can see that we hit our first breakpoint at OpenProcess:



```
RIP ──────→ 00007FFC88D1ADE0  - 48:FF25 896E0600   jmp qword ptr ds:[<OpenProcess>]   OpenProcess
            00007FFC88D1ADE7    CC                 int3
```

And if we step over, we go to the OpenProcess, and we if see the 3rd parameter, we will get the pid

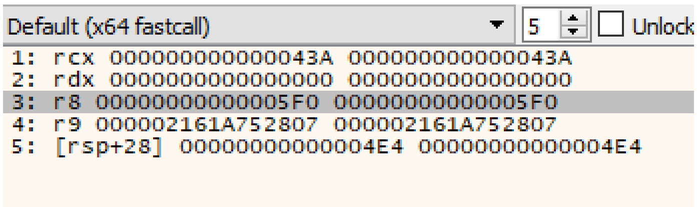


```
RIP ──→ 00007FFC86ADC0F0   4C:8BDC              mov r11,rsp                              OpenProcess
        00007FFC86ADC0F3   48:83EC 68           sub rsp,68
        00007FFC86ADC0F7   49:8363 C0 00        and qword ptr ds:[r11-40],0
        00007FFC86ADC0FC   4D:8D4B B8           lea r9,qword ptr ds:[r11-48]
        00007FFC86ADC100   49:63C0              movsxd rax,r8d
        00007FFC86ADC103   0F57C0               xorps xmm0,xmm0
        00007FFC86ADC106   C74424 30 30000000   mov dword ptr ss:[rsp+30],30             30:'0'
        00007FFC86ADC10E   4D:8D43 C8           lea r8,qword ptr ds:[r11-38]             [r11-38]:SdbFindFirstTag+13D
        00007FFC86ADC112   49:8363 D0 00        and qword ptr ds:[r11-30],0
        00007FFC86ADC117   F7DA                 neg edx
        00007FFC86ADC119   49:8943 B8           mov qword ptr ds:[r11-48],rax
        00007FFC86ADC11D   8BD1                 mov edx,ecx
        00007FFC86ADC11F   49:8D4B 20           lea rcx,qword ptr ds:[r11+20]
        00007FFC86ADC123   1BC0                 sbb eax,eax
        00007FFC86ADC125   83E0 02              and eax,2
        00007FFC86ADC128   894424 48            mov dword ptr ss:[rsp+48],eax
        00007FFC86ADC12C   49:8363 D8 00        and qword ptr ds:[r11-28],0
        00007FFC86ADC131   F3:0F7F4424 50       movdqu xmmword ptr ss:[rsp+50],xmm0
        00007FFC86ADC137   48:FF15 3A7E1900     call qword ptr ds:[<NtOpenProcess>]
        00007FFC86ADC13E   0F1F4400 00          nop dword ptr ds:[rax+rax],eax
        00007FFC86ADC143   85C0                 test eax,eax
        00007FFC86ADC145 ⌄ 78 0E                js kernelbase.7FFC86ADC155
        00007FFC86ADC147   48:8B8424 88000000   mov rax,qword ptr ss:[rsp+88]
        00007FFC86ADC14F   48:83C4 68           add rsp,68
        00007FFC86ADC153   C3                   ret
        00007FFC86ADC154   CC                   int3
        00007FFC86ADC155   8BC8                 mov ecx,eax
        00007FFC86ADC157   E8 14F2FFFF          call kernelbase.7FFC86ADB370
        00007FFC86ADC15C   33C0                 xor eax,eax
        00007FFC86ADC15E ^ EB EF                jmp kernelbase.7FFC86ADC14F
        00007FFC86ADC160   CC                   int3
        00007FFC86ADC161   CC                   int3
        00007FFC86ADC162   CC                   int3
        00007FFC86ADC163   CC                   int3
        00007FFC86ADC164   CC                   int3
        00007FFC86ADC165   CC                   int3
        00007FFC86ADC166   CC                   int3
        00007FFC86ADC167   CC                   int3
        00007FFC86ADC168   CC                   int3
        00007FFC86ADC169   CC                   int3
        00007FFC86ADC16A   CC                   int3
        00007FFC86ADC16B   CC                   int3
```

Because it takes pid as the 3rd parameter:

```C++
HANDLE OpenProcess(
  [in] DWORD dwDesiredAccess,
  [in] BOOL  bInheritHandle,
  [in] DWORD dwProcessId
);
```

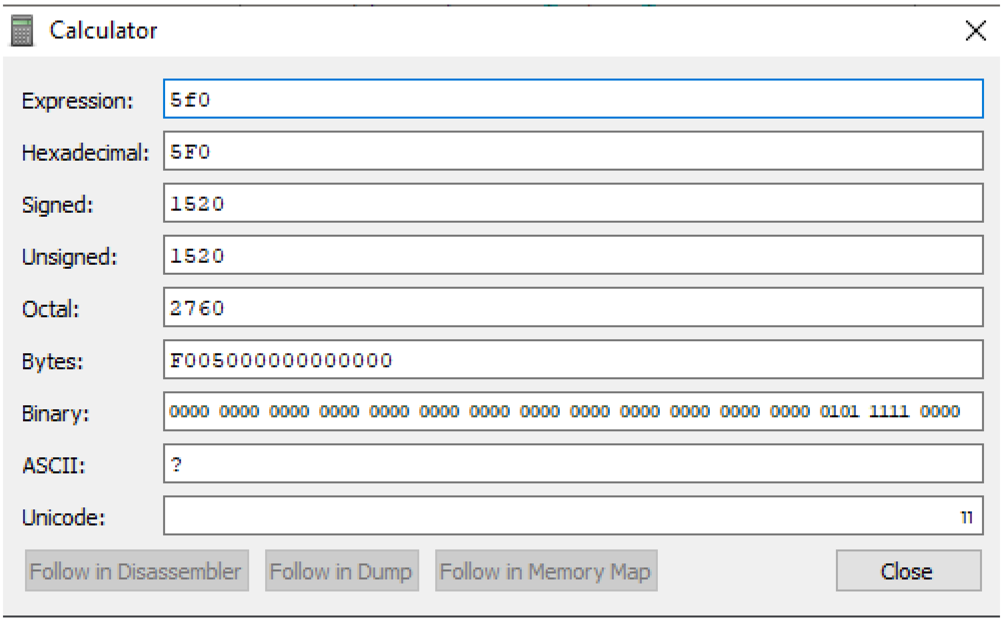
Here’s the 3rd parameter:



Default (x64 fastcall) ▾ 5 □ Unlock

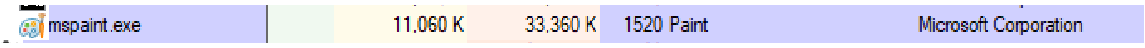
```
1: rcx 000000000000043A 000000000000043A
2: rdx 0000000000000000 0000000000000000
3: r8 00000000000005F0 00000000000005F0
4: r9 000002161A752807 000002161A752807
5: [rsp+28] 00000000000004E4 00000000000004E4
```

In the calculator we can see the pid number:



Expression: 5f0
Hexadecimal: 5F0
Signed: 1520
Unsigned: 1520
Octal: 2760
Bytes: F005000000000000
Binary: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0101 1111 0000
ASCII: ?
Unicode: ᗰ

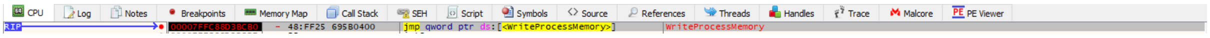
We can compare it here:



mspaint.exe		11,060 K	33,360 K	1520 Paint	Microsoft Corporation

So, as we can see we confirm that the process it is trying to access is mspaint, and now it will allocate some memory for the shellcode, and then it will be filled with the shellcode.

Now if we run the program, it will hit the breakpoint at: WriteProcessMemory



```
RIP ──────> 00007FFC88D3BCB0  - 48:FF25 695B0400    jmp qword ptr ds:[<WriteProcessMemory>]    WriteProcessMemory
```

Now if we follow the second parameter here, we will get the address where the shellcode is going to be injected, so we get this address, and now follow it in the process hacker, you can see that the shellcode has been injected into this memory, so now we can extract the shellcode, and can compare it with the original shellcode which we have.