Required Tools

Introduction

Before beginning the malware development journey, it is necessary to prepare the development workspace by installing malware development and reverse engineering tools. These tools will aid one in the development and analysis of the malware and will be used throughout the modules.

Reverse Engineering Tools

Several of the tools mentioned focus more on reverse engineering rather than development. It is essential to reverse engineer the malware built to fully understand its internal workings and have an understanding of what the malware analysts will see upon inspecting the malware.

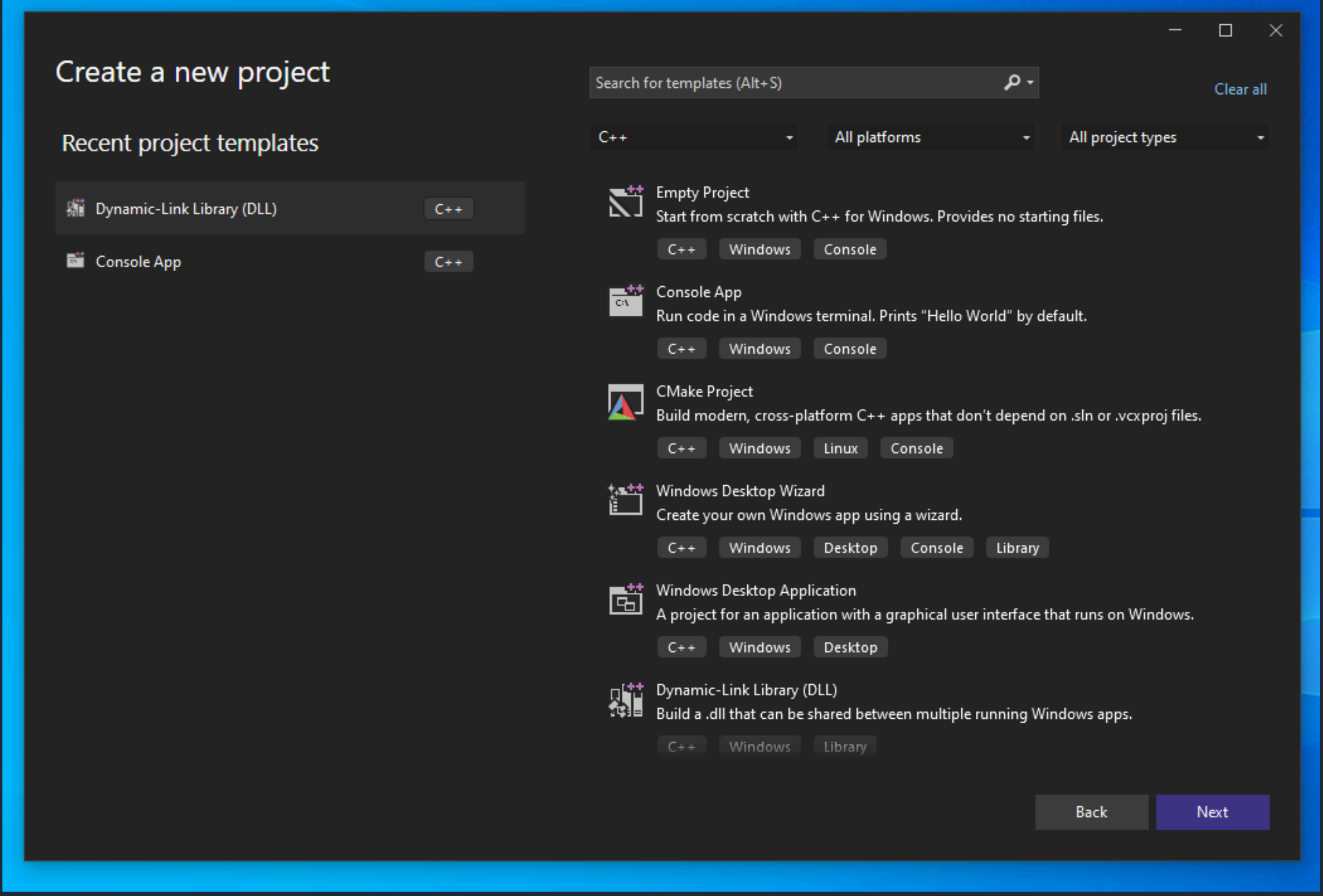
Tools To Install

Install the following tools:

- Visual Studio - This is the development environment where the coding & compiling process will occur. Install the C/C++ Runtime.
- x64dbg - x64dbg is a debugger that will be used throughout the modules to get an internal understanding of the developed malware.
- PE-Bear - PE-bear is a multiplatform reversing tool for PE files. It will also be used to assess the developed malware and look for suspicious indicators.
- Process Hacker 2 - Process Hacker is a powerful, multi-purpose tool that helps monitor system resources, debug software and detect malware.
- Msfvenom - Msfvenom is a command line interface tool that is used to create, manipulate, and output payloads.

Visual Studio

Visual Studio is an integrated development environment (IDE) developed by Microsoft. It is used to develop a wide array of software such as web applications, web services and computer programs. It also comes with development and debugging tools for building and testing applications. Visual Studio will be the main IDE used for development in this course.



Create a new project

Recent project templates

- Dynamic-Link Library (DLL) — C++
- Console App — C++

Search for templates (Alt+S)

Clear all

C++ | All platforms | All project types

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ · Windows · Console

Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ · Windows · Console

CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ · Windows · Linux · Console

Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ · Windows · Desktop · Console · Library

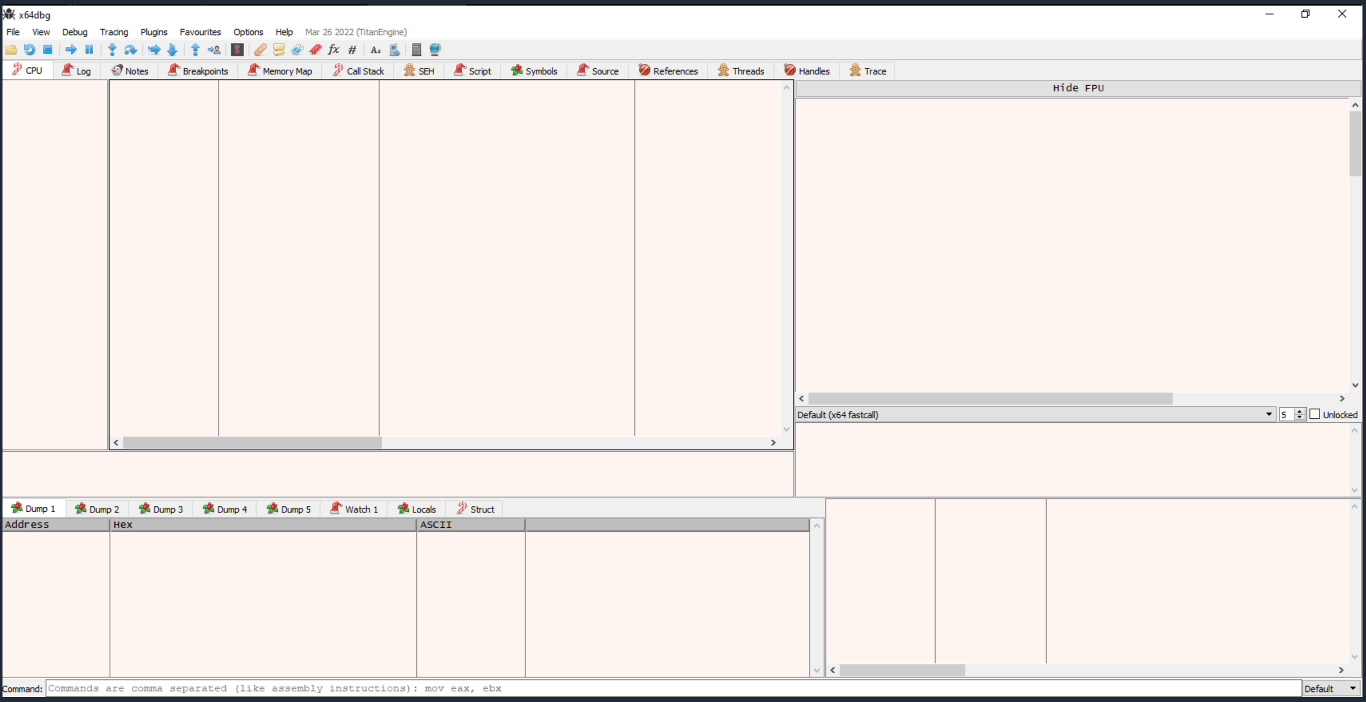
Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.
C++ · Windows · Desktop

Dynamic-Link Library (DLL)
Build a .dll that can be shared between multiple running Windows apps.
C++ · Windows · Library

Back | Next

x64dbg

x64dbg is an open-source debugging utility for x64 and x86 Windows binaries. It is used to analyze and debug user-mode applications and kernel-mode drivers. It provides a graphical user interface that allows users to inspect and analyze the state of their programs and view memory contents, assembly instructions, and register values. With x64dbg, users can set breakpoints, view stack and heap data, step through code, and read and write memory values.


x64dbg
File View Debug Tracing Plugins Favourites Options Help Mar 26 2022 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Hide FPU
Default (x64 fastcall)
5
Unlocked
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Address
Hex
ASCII
Command:
Default

```
Commands are comma separated (like assembly instructions): mov eax, ebx
```

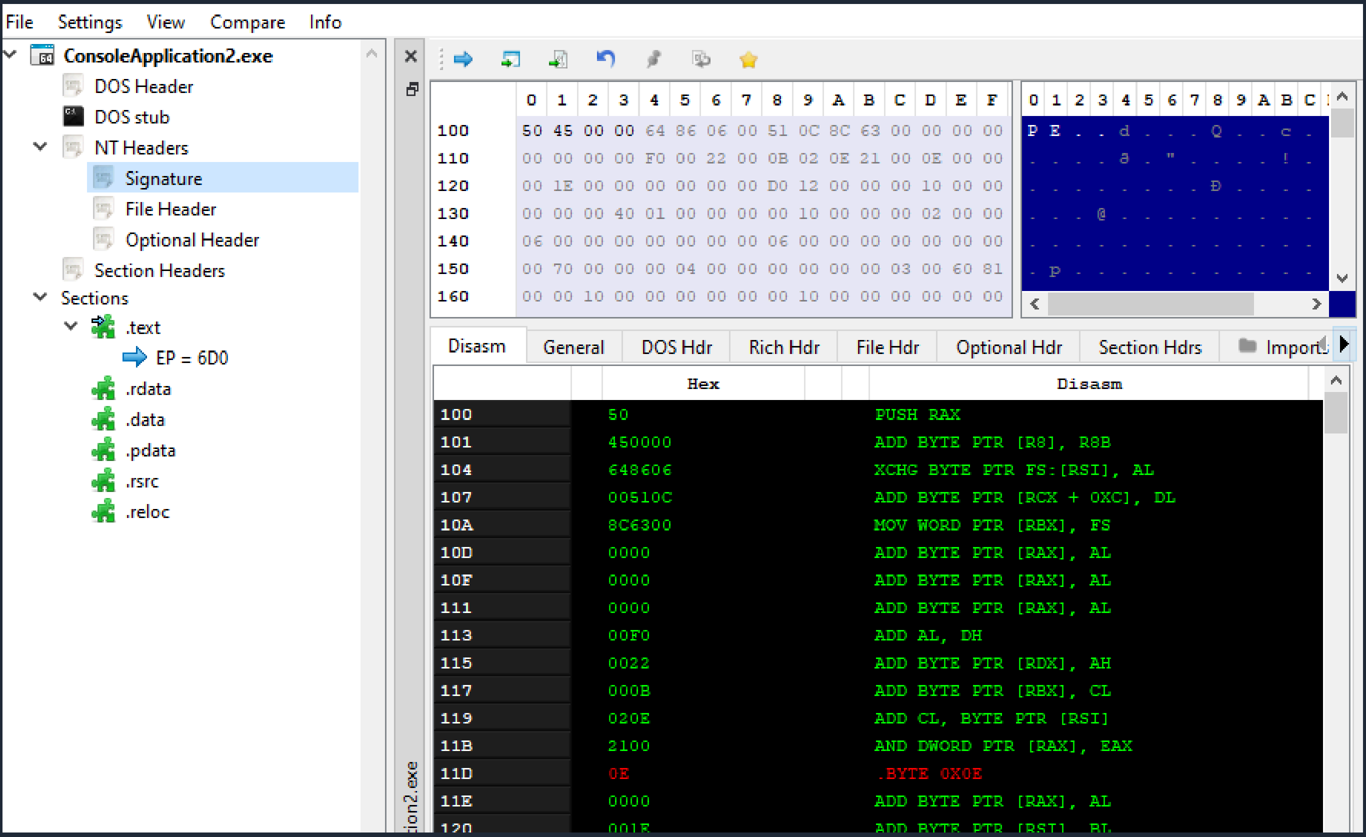
The main "CPU" tab has 4 screens:

1. Disassembly (Top-left): This window displays the assembly instructions being executed by the application.
2. Dump (Bottom-left): This window displays the memory contents of the application being debugged.
3. Registers (Top-right): This window displays the values of the CPU registers.
4. Stack (Bottom-right): This window displays the contents of the stack.

The remaining tabs also provide useful information but they will be discussed in the modules when they are used.

PE-Bear

PE-Bear is a free, open-source tool designed to help malware analysts and reverse engineers quickly and easily analyze Windows Portable Executable (PE) files. It helps to analyze and visualize the structure of the PE file, view the imports and exports of each module, and perform static analysis to detect anomalies and possible malicious code. PE-bear also includes features such as PE header and section validation, as well as a hex editor.



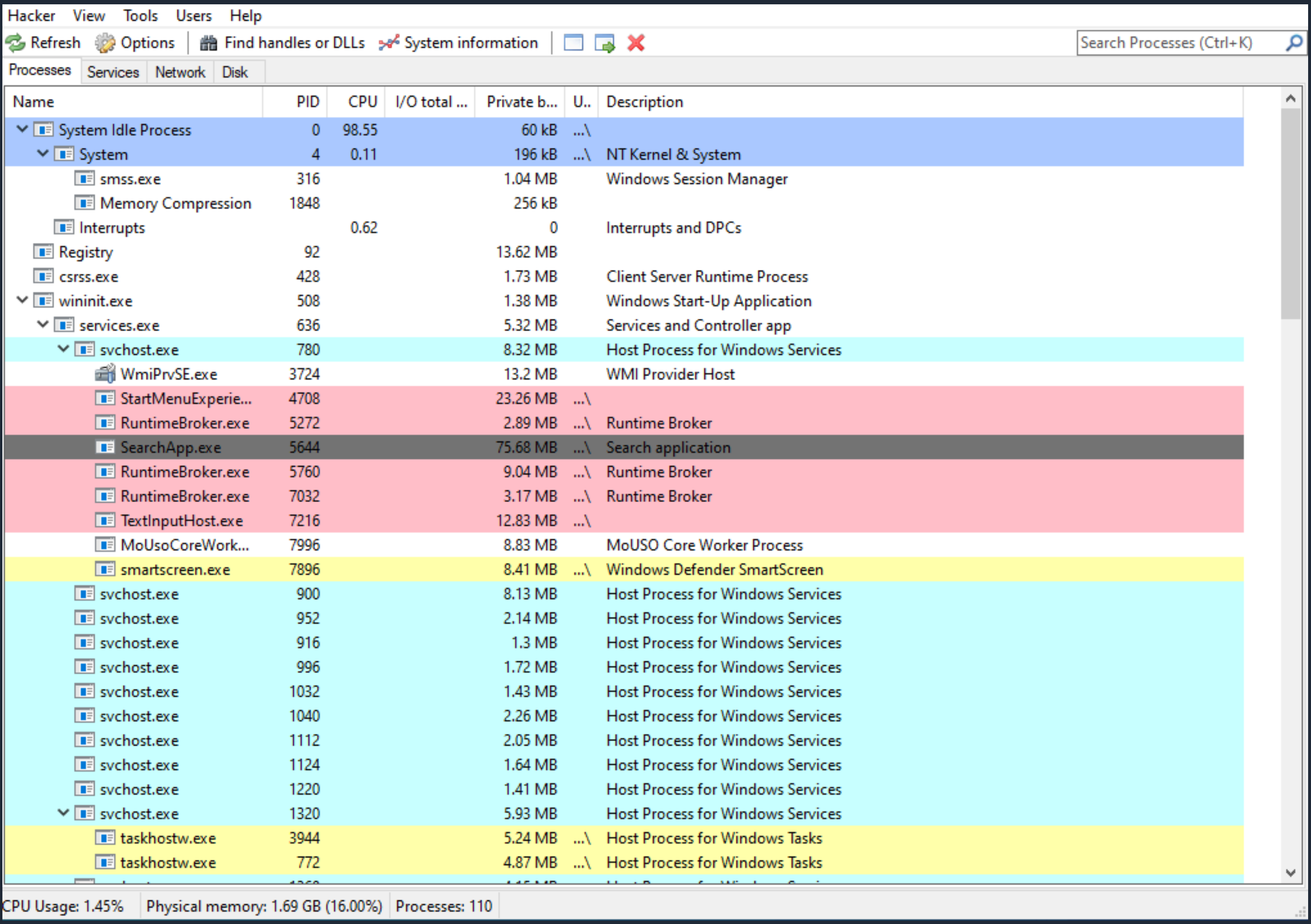
```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100  50 45 00 00 64 86 06 00 51 0C 8C 63 00 00 00 00
110  00 00 00 00 F0 00 22 00 0B 02 0E 21 00 0E 00 00
120  00 1E 00 00 00 00 00 00 D0 12 00 00 00 10 00 00
130  00 00 00 40 01 00 00 00 00 10 00 00 00 02 00 00
140  06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00
150  00 70 00 00 00 04 00 00 00 00 00 00 03 00 60 81
160  00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00
```

```
0 1 2 3 4 5 6 7 8 9 A B C
P E . . d . . . Q . . c .
. . . . à . " . . . . ! .
. . . . . . . . Đ . . . .
. . . @ . . . . . . . . .
. . . . . . . . . . . . .
. p . . . . . . . . . . .
```

```
      Hex        Disasm
100   50         PUSH RAX
101   450000     ADD BYTE PTR [R8], R8B
104   648606     XCHG BYTE PTR FS:[RSI], AL
107   00510C     ADD BYTE PTR [RCX + 0XC], DL
10A   8C6300     MOV WORD PTR [RBX], FS
10D   0000       ADD BYTE PTR [RAX], AL
10F   0000       ADD BYTE PTR [RAX], AL
111   0000       ADD BYTE PTR [RAX], AL
113   00F0       ADD AL, DH
115   0022       ADD BYTE PTR [RDX], AH
117   000B       ADD BYTE PTR [RBX], CL
119   020E       ADD CL, BYTE PTR [RSI]
11B   2100       AND DWORD PTR [RAX], EAX
11D   0E         .BYTE 0X0E
11E   0000       ADD BYTE PTR [RAX], AL
120   001E       ADD BYTE PTR [RSI], BL
```

Process Hacker

Process Hacker is an open-source tool for viewing and manipulating processes and services on Windows. It is similar to Task Manager but provides more information and advanced features. It can be used to terminate processes and services, view detailed process information and statistics, set process priorities and more. Process Hacker will be useful when analyzing running processes to view items such as loaded DLLs and memory regions.

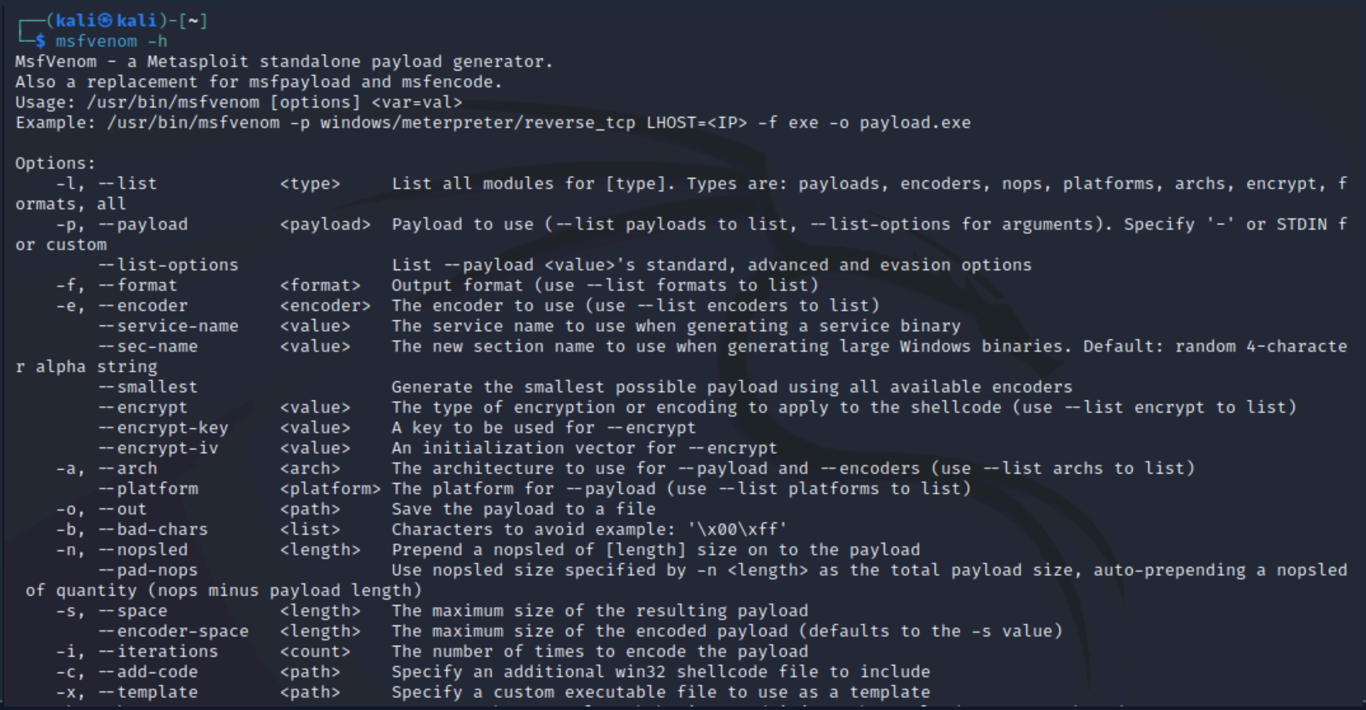


Name	PID	CPU	I/O total ...	Private b...	U..	Description
System Idle Process	0	98.55		60 kB	...\	
System	4	0.11		196 kB	...\	NT Kernel & System
smss.exe	316			1.04 MB		Windows Session Manager
Memory Compression	1848			256 kB		
Interrupts		0.62		0		Interrupts and DPCs
Registry	92			13.62 MB		
csrss.exe	428			1.73 MB		Client Server Runtime Process
wininit.exe	508			1.38 MB		Windows Start-Up Application
services.exe	636			5.32 MB		Services and Controller app
svchost.exe	780			8.32 MB		Host Process for Windows Services
WmiPrvSE.exe	3724			13.2 MB		WMI Provider Host
StartMenuExperie...	4708			23.26 MB	...\	
RuntimeBroker.exe	5272			2.89 MB	...\	Runtime Broker
SearchApp.exe	5644			75.68 MB	...\	Search application
RuntimeBroker.exe	5760			9.04 MB	...\	Runtime Broker
RuntimeBroker.exe	7032			3.17 MB	...\	Runtime Broker
TextInputHost.exe	7216			12.83 MB	...\	
MoUsoCoreWork...	7996			8.83 MB		MoUSO Core Worker Process
smartscreen.exe	7896			8.41 MB	...\	Windows Defender SmartScreen
svchost.exe	900			8.13 MB		Host Process for Windows Services
svchost.exe	952			2.14 MB		Host Process for Windows Services
svchost.exe	916			1.3 MB		Host Process for Windows Services
svchost.exe	996			1.72 MB		Host Process for Windows Services
svchost.exe	1032			1.43 MB		Host Process for Windows Services
svchost.exe	1040			2.26 MB		Host Process for Windows Services
svchost.exe	1112			2.05 MB		Host Process for Windows Services
svchost.exe	1124			1.64 MB		Host Process for Windows Services
svchost.exe	1220			1.41 MB		Host Process for Windows Services
svchost.exe	1320			5.93 MB		Host Process for Windows Services
taskhostw.exe	3944			5.24 MB	...\	Host Process for Windows Tasks
taskhostw.exe	772			4.87 MB	...\	Host Process for Windows Tasks

CPU Usage: 1.45% Physical memory: 1.69 GB (16.00%) Processes: 110

Msfvenom

Msfvenom is a Metasploit framework standalone payload generator that allows users to generate various types of payloads. These payloads will be used by the malware created in this course.



```
┌──(kali㉿kali)-[~]
└─$ msfvenom -h
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>
Example: /usr/bin/msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe

Options:
    -l, --list            <type>     List all modules for [type]. Types are: payloads, encoders, nops, platforms, archs, encrypt, f
ormats, all
    -p, --payload         <payload>  Payload to use (--list payloads to list, --list-options for arguments). Specify '-' or STDIN f
or custom
        --list-options               List --payload <value>'s standard, advanced and evasion options
    -f, --format          <format>   Output format (use --list formats to list)
    -e, --encoder         <encoder>  The encoder to use (use --list encoders to list)
        --service-name    <value>    The service name to use when generating a service binary
        --sec-name        <value>    The new section name to use when generating large Windows binaries. Default: random 4-characte
r alpha string
        --smallest                   Generate the smallest possible payload using all available encoders
        --encrypt         <value>    The type of encryption or encoding to apply to the shellcode (use --list encrypt to list)
        --encrypt-key     <value>    A key to be used for --encrypt
        --encrypt-iv      <value>    An initialization vector for --encrypt
    -a, --arch            <arch>     The architecture to use for --payload and --encoders (use --list archs to list)
        --platform        <platform> The platform for --payload (use --list platforms to list)
    -o, --out             <path>     Save the payload to a file
    -b, --bad-chars       <list>     Characters to avoid example: '\x00\xff'
    -n, --nopsled         <length>   Prepend a nopsled of [length] size on to the payload
        --pad-nops                   Use nopsled size specified by -n <length> as the total payload size, auto-prepending a nopsled
 of quantity (nops minus payload length)
    -s, --space           <length>   The maximum size of the resulting payload
        --encoder-space   <length>   The maximum size of the encoded payload (defaults to the -s value)
    -i, --iterations      <count>    The number of times to encode the payload
    -c, --add-code        <path>     Specify an additional win32 shellcode file to include
    -x, --template        <path>     Specify a custom executable file to use as a template
```

Previous Modules Undo Next