2023

Usman Sikander

Offensive security researcher

[medium.com/@merasor07](mailto:medium.com/@merasor07)

linkedin.com/in/usman-sikander13/

11/23/2023

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



Contents

[Introduction 2](#_Toc151651331)

[Capabilities 2](#_Toc151651332)

[Technical Details and Chain flow 3](#_Toc151651333)

[File Info: 3](#_Toc151651334)

[SAMSAM also known as by security vendors: 3](#_Toc151651335)

[Flow of attack and execution: 3](#_Toc151651336)

[Tools and Environment 4](#_Toc151651337)

[Stage 1 (WinDir.exe) 4](#_Toc151651338)

[Basic and Advanced Static Analysis 4](#_Toc151651339)

[Basic Information 4](#_Toc151651340)

[Packing 8](#_Toc151651341)

[Detect-It-Easy 8](#_Toc151651342)

[Capa-Output 8](#_Toc151651343)

[Static Analysis 9](#_Toc151651344)

[Basic Dynamic Analysis 10](#_Toc151651345)

[Procmon and Process Hacker 10](#_Toc151651346)

[Advanced Dynamic Analysis 12](#_Toc151651347)

[Breakpoints: 12](#_Toc151651348)

[Extracted TTP’s 24](#_Toc151651349)

[MITRE ATT&CK MAPPING 24](#_Toc151651350)

[Recreation and Security controls validation 24](#_Toc151651351)

[Mitigation 26](#_Toc151651352)

[YARA 26](#_Toc151651353)

[Conclusion 27](#_Toc151651354)

# 

# 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

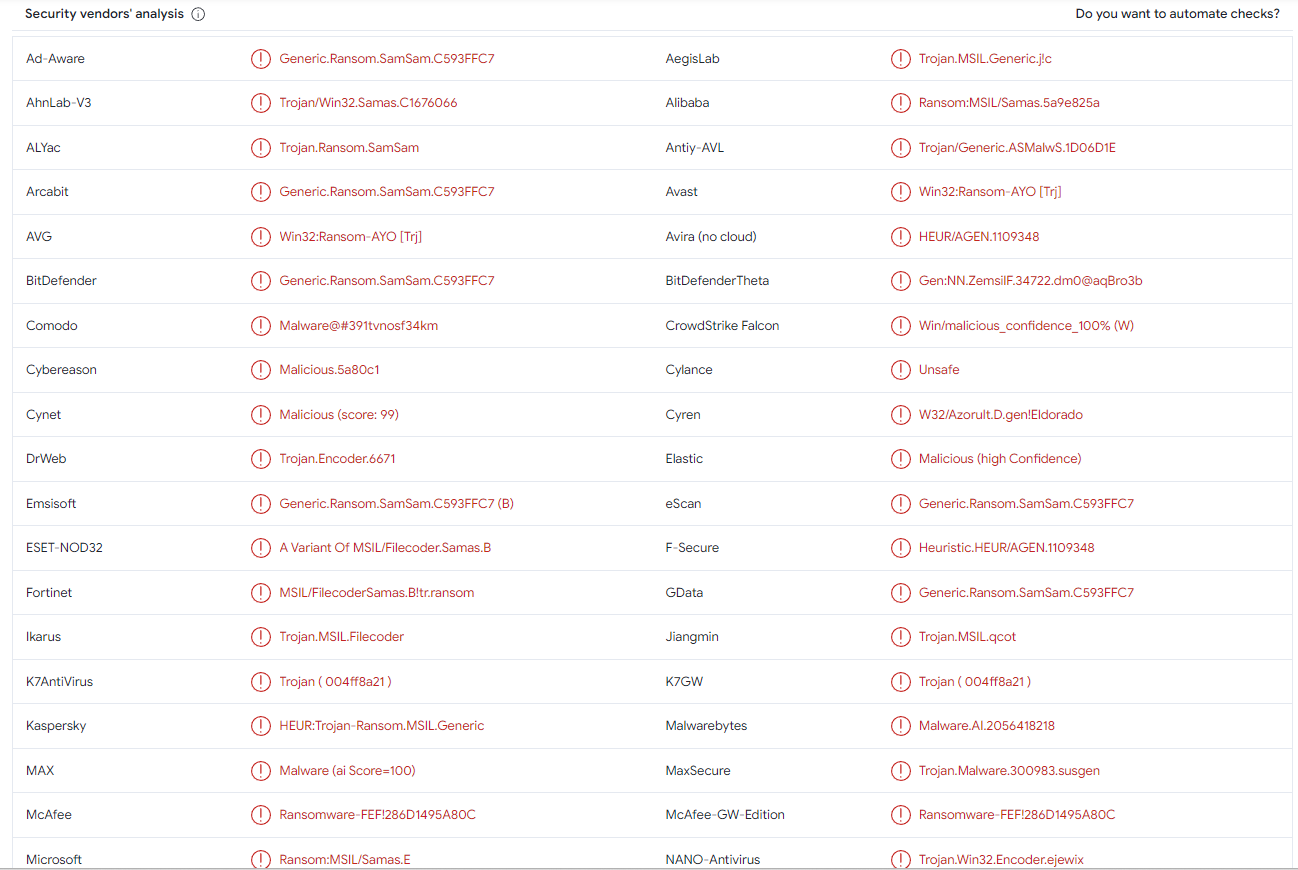
* Samsam creates files inside the user directory
* Samsam queries all running process
* Samsam perform discovery and queries GUID
* Samsam utilizes the defense evasion technique Masquerading
* Samsam deletes the backup files
* Samsam encrypts the entire system using RSA-2048
* Creates guard pages, often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

# Technical Details and Chain flow

## File Info:



## SAMSAM also known as by security vendors:

******

## 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\\_SALTEAAAAPl2nDcnZ1T3UwfsFQMZjR7XzfWagWbfl2fYA3WAGh4tEAAAAH8waUVAF3EUA/8505vkPNOOgEMLO1t6CJvcGT6nAesg5YS2M4bKhDkWUKn9catLXAcFPEx0jYQbRX7gyYQxI9M=</html><body style='background-color:lightgrey;'><pre><font color='Red'><center><h3>&#35;&#87;&#104;&#97;&#116;&#32;&#104;&#97;&#112;&#112;&#101;&#110;&#101;&#100;&#32;&#116;&#111;&#32;&#121;&#111;&#117;&#114;&#32;&#102;&#105;&#108;&#101;&#115;&#63;</h3></center></font>&#65;&#108;&#108;&#32;&#121;&#111;&#117;&#114;&#32;&#102;&#105;&#108;&#101;&#115;&#32;&#101;&#110;&#99;&#114;&#121;&#112;&#116;&#101;&#100;&#32;&#119;&#105;&#116;&#104;&#32;&#82;&#83;&#65;&#45;&#50;&#48;&#52;&#56;&#32;&#101;&#110;&#99;&#114;&#121;&#112;&#116;&#105;&#111;&#110;&#44;&#32;&#70;&#111;&#114;&#32;&#109;&#111;&#114;&#101;&#32;&#105;&#110;&#102;&#111;&#114;&#109;&#97;&#116;&#105;&#111;&#110;&#32;&#115;&#101;&#97;&#114;&#99;&#104;&#32;&#105;&#110;&#32;&#71;&#111;&#111;&#103;&#108;&#101;&#32;&#34;&#82;&#83;&#65;&#32;&#69;&#110;&#99;&#114;&#121;&#112;&#116;&#105;&#111;&#110;&#34;<font color='Red'><center><h3>&#35;&#72;&#111;&#119;&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#111;&#118;&#101;&#114;&#32;&#102;&#105;&#108;&#101;&#115;&#63;</h3></center></font>&#82;&#83;&#65;&#32;&#105;&#115;&#32;&#97;&#32;&#97;&#115;&#121;&#109;&#109;&#101;&#116;&#114;&#105;&#99;&#32;&#99;&#114;&#121;&#112;&#116;&#111;&#103;&#114;&#97;&#112;&#104;&#105;&#99;&#32;&#97;&#108;&#103;&#111;&#114;&#105;&#116;&#104;&#109;&#44;&#32;&#89;&#111;&#117;&#32;&#110;&#101;&#101;&#100;&#32;&#111;&#110;&#101;&#32;&#107;&#101;&#121;&#32;&#102;&#111;&#114;&#32;&#101;&#110;&#99;&#114;&#121;&#112;&#116;&#105;&#111;&#110;&#32;&#97;&#110;&#100;&#32;&#111;&#110;&#101;&#32;&#107;&#101;&#121;&#32;&#102;&#111;&#114;&#32;&#100;&#101;&#99;&#114;&#121;&#112;&#116;&#105;&#111;&#110;&#10;&#83;&#111;&#32;&#121;&#111;&#117;&#32;&#110;&#101;&#101;&#100;&#32;&#80;&#114;&#105;&#118;&#97;&#116;&#101;&#32;&#107;&#101;&#121;&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#111;&#118;&#101;&#114;&#32;&#121;&#111;&#117;&#114;&#32;&#102;&#105;&#108;&#101;&#115;&#46;&#10;&#73;&#116;&#39;&#115;&#32;&#110;&#111;&#116;&#32;&#112;&#111;&#115;&#115;&#105;&#98;&#108;&#101;&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#111;&#118;&#101;&#114;&#32;&#121;&#111;&#117;&#114;&#32;&#102;&#105;&#108;&#101;&#115;&#32;&#119;&#105;&#116;&#104;&#111;&#117;&#116;&#32;&#112;&#114;&#105;&#118;&#97;&#116;&#101;&#32;&#107;&#101;&#121;<font color='red'><center><h3>&#35;&#72;&#111;&#119;&#32;&#116;&#111;&#32;&#103;&#101;&#116;&#32;&#112;&#114;&#105;&#118;&#97;&#116;&#101;&#32;&#107;&#101;&#121;&#63;</h3></center></font>&#89;&#111;&#117;&#32;&#99;&#97;&#110;&#32;&#103;&#101;&#116;&#32;&#121;&#111;&#117;&#114;&#32;&#112;&#114;&#105;&#118;&#97;&#116;&#101;&#32;&#107;&#101;&#121;&#32;&#105;&#110;&#32;&#51;&#32;&#101;&#97;&#115;&#121;&#32;&#115;&#116;&#101;&#112;&#58;<br><font color='DrakRed'>&#83;&#116;&#101;&#112;&#49;&#58;</font>&#32;&#89;&#111;&#117;&#32;&#109;&#117;&#115;&#116;&#32;&#115;&#101;&#110;&#100;&#32;&#117;&#115;&#32;<font color='red'>&#32;&#66;&#105;&#116;&#67;&#111;&#105;&#110;&#115;</font>&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#101;&#105;&#118;&#101;&#32;&#65;&#76;&#76;&#32;&#80;&#114;&#105;&#118;&#97;&#116;&#101;&#32;&#75;&#101;&#121;&#115;&#32;&#102;&#111;&#114;&#32;&#65;&#76;&#76;&#32;&#97;&#102;&#102;&#101;&#99;&#116;&#101;&#100;&#32;&#80;&#67;&#39;&#115;&#46;<br><font color='DrakRed'>&#83;&#116;&#101;&#112;&#50;&#58;</font>&#32;&#65;&#102;&#116;&#101;&#114;&#32;&#121;&#111;&#117;&#32;&#115;&#101;&#110;&#100;&#32;&#117;&#115;&#32;<font color='red'><br><br><br><br><font color='DrakRed'>&#83;&#116;&#101;&#112;&#51;&#58;</font> &#87;&#101;&#32;&#119;&#105;&#108;&#108;&#32;&#114;&#101;&#112;&#108;&#121;&#32;&#116;&#111;&#32;&#121;&#111;&#117;&#114;&#32;&#99;&#111;&#109;&#109;&#101;&#110;&#116;&#32;&#119;&#105;&#116;&#104;&#32;&#97;&#32;&#100;&#101;&#99;&#114;&#121;&#112;&#116;&#105;&#111;&#110;&#32;&#115;&#111;&#102;&#116;&#119;&#97;&#114;&#101;&#44;&#32;&#89;&#111;&#117;&#32;&#115;&#104;&#111;&#117;&#108;&#100;&#32;&#114;&#117;&#110;&#32;&#105;&#116;&#32;&#111;&#110;&#32;&#121;&#111;&#117;&#114;&#32;&#97;&#102;&#102;&#101;&#99;&#116;&#101;&#100;&#32;&#80;&#67;&#32;&#97;&#110;&#100;&#32;&#97;&#108;&#108;&#32;&#101;&#110;&#99;&#114;&#121;&#112;&#116;&#101;&#100;&#32;&#102;&#105;&#108;&#101;&#115;&#32;&#119;&#105;&#108;&#108;&#32;&#98;&#101;&#32;&#114;&#101;&#99;&#111;&#118;&#101;&#114;&#101;&#100;<br><font color='DrakRed'>\*</font>&#79;&#117;&#114;&#32;&#83;&#105;&#116;&#101;&#32;&#65;&#100;&#100;&#114;&#101;&#115;&#115;&#58;<a href=''><b></b></font><br>&#40;&#73;&#102;&#32;&#121;&#111;&#117;&#32;&#115;&#101;&#110;&#100;&#32;&#117;&#115;&#32;<font color='red'>&#32;&#66;&#105;&#116;&#99;&#111;&#105;&#110;&#115;&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#101;&#105;&#118;&#101;&#32;&#104;&#97;&#108;&#102;&#32;&#111;&#102;&#32;&#107;&#101;&#121;&#115;&#40;&#114;&#97;&#110;&#100;&#111;&#109;&#108;&#121;&#41;&#32;&#97;&#110;&#100;&#32;&#97;&#102;&#116;&#101;&#114;&#32;&#121;&#111;&#117;&#32;&#118;&#101;&#114;&#105;&#102;&#121;&#32;&#105;&#116;&#32;&#115;&#101;&#110;&#100;&#32;&#50;&#110;&#100;&#32;&#104;&#97;&#108;&#102;&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#101;&#105;&#118;&#101;&#32;&#97;&#108;&#108;&#32;&#107;&#101;&#121;&#115;&#32;&#41;<br> <font color='red'><center><h3>&#72;&#111;&#119;&#32;&#84;&#111;&#32;&#65;&#99;&#99;&#101;&#115;&#115;&#32;&#84;&#111;&#32;&#79;&#117;&#114;&#32;&#83;&#105;&#116;&#101;</h3></center></font>&#70;&#111;&#114;&#32;&#97;&#99;&#99;&#101;&#115;&#115;&#32;&#116;&#111;&#32;&#111;&#117;&#114;&#32;&#115;&#105;&#116;&#101;&#32;&#121;&#111;&#117;&#32;&#109;&#117;&#115;&#116;&#32;&#105;&#110;&#115;&#116;&#97;&#108;&#108;&#32;&#84;&#111;&#114;&#32;&#98;&#114;&#111;&#119;&#115;&#101;&#114;&#32;&#97;&#110;&#100;&#32;&#101;&#110;&#116;&#101;&#114;&#32;&#111;&#117;&#114;&#32;&#115;&#105;&#116;&#101;&#32;&#85;&#82;&#76;&#32;&#105;&#110;&#32;&#121;&#111;&#117;&#114;&#32;&#116;&#111;&#114;&#32;&#98;&#114;&#111;&#119;&#115;&#101;&#114;&#46;&#10;&#89;&#111;&#117;&#32;&#99;&#97;&#110;&#32;&#100;&#111;&#119;&#110;&#108;&#111;&#97;&#100;&#32;&#116;&#111;&#114;&#32;&#98;&#114;&#111;&#119;&#115;&#101;&#114;&#32;&#102;&#114;&#111;&#109;&#32;<font color='DrakRed'></font><a href='https://www.torproject.org/download/download.html.en'>https://www.torproject.org/download/download.html.en</a>&#70;&#111;&#114;&#32;&#109;&#111;&#114;&#101;&#32;&#105;&#110;&#102;&#111;&#114;&#109;&#97;&#116;&#105;&#111;&#110;&#32;&#112;&#108;&#101;&#97;&#115;&#101;&#32;&#115;&#101;&#97;&#114;&#99;&#104;&#32;&#105;&#110;&#32;&#71;&#111;&#111;&#103;&#108;&#101;&#32;&#34;&#72;&#111;&#119;&#32;&#116;&#111;&#32;&#97;&#99;&#99;&#101;&#115;&#115;&#32;&#111;&#110;&#105;&#111;&#110;&#32;&#115;&#105;&#116;&#101;&#115;&#34;&#32;<br><br><font color='red'><center><h3>&#35;&#32;&#84;&#101;&#115;&#116;&#32;&#68;&#101;&#99;&#114;&#121;&#112;&#116;&#105;&#111;&#110;&#32;&#35;</h3></center></font><br>&#67;&#104;&#101;&#99;&#107;&#32;&#111;&#117;&#114;&#32;&#115;&#105;&#116;&#101;&#44;&#32;&#89;&#111;&#117;&#32;&#99;&#97;&#110;&#32;&#117;&#112;&#108;&#111;&#97;&#100;&#32;&#50;&#32;&#101;&#110;&#99;&#114;&#121;&#112;&#116;&#101;&#100;&#32;&#102;&#105;&#108;&#101;&#115;&#32;&#97;&#110;&#100;&#32;&#119;&#101;&#32;&#119;&#105;&#108;&#108;&#32;&#100;&#101;&#99;&#114;&#121;&#112;&#116;&#32;&#121;&#111;&#117;&#114;&#32;&#102;&#105;&#108;&#101;&#115;&#32;&#97;&#115;&#32;&#100;&#101;&#109;&#111;&#46;&#32;&#32;<br><font color='red'><center><h3>&#35;&#87;&#104;&#101;&#114;&#101;&#32;&#116;&#111;&#32;&#98;&#117;&#121;&#32;&#66;&#105;&#116;&#99;&#111;&#105;&#110;</h3></center></font><br>&#87;&#101;&#32;&#97;&#100;&#118;&#105;&#99;&#101;&#32;&#121;&#111;&#117;&#32;&#116;&#111;&#32;&#98;&#117;&#121;&#32;&#66;&#105;&#116;&#99;&#111;&#105;&#110;&#32;&#119;&#105;&#116;&#104;&#32;&#67;&#97;&#115;&#104;&#32;&#68;&#101;&#112;&#111;&#115;&#105;&#116;&#32;&#111;&#114;&#32;&#87;&#101;&#115;&#116;&#101;&#114;&#110;&#85;&#110;&#105;&#111;&#110;&#32;&#70;&#114;&#111;&#109;&#32;&#104;&#116;&#116;&#112;&#115;&#58;&#47;&#47;&#108;&#111;&#99;&#97;&#108;&#98;&#105;&#116;&#99;&#111;&#105;&#110;&#115;&#46;&#99;&#111;&#109;&#47;&#32;&#111;&#114;&#32;&#104;&#116;&#116;&#112;&#115;&#58;&#47;&#47;&#99;&#111;&#105;&#110;&#99;&#97;&#102;&#101;&#46;&#99;&#111;&#109;&#47;&#98;&#117;&#121;&#98;&#105;&#116;&#99;&#111;&#105;&#110;&#115;&#119;&#101;&#115;&#116;&#101;&#114;&#110;&#46;&#112;&#104;&#112;&#10;&#66;&#101;&#99;&#97;&#117;&#115;&#101;&#32;&#116;&#104;&#101;&#121;&#32;&#100;&#111;&#110;&#39;&#116;&#32;&#110;&#101;&#101;&#100;&#32;&#97;&#110;&#121;&#32;&#118;&#101;&#114;&#105;&#102;&#105;&#99;&#97;&#116;&#105;&#111;&#110;&#32;&#97;&#110;&#100;&#32;&#115;&#101;&#110;&#100;&#32;&#121;&#111;&#117;&#114;&#32;&#66;&#105;&#116;&#99;&#111;&#105;&#110;&#32;&#113;&#117;&#105;&#99;&#107;&#108;&#121;&#46;<br><br><font color='red'><center><h3>&#35;&#100;&#101;&#97;&#100;&#108;&#105;&#110;&#101;</h3></center></font><br>&#89;&#111;&#117;&#32;&#106;&#117;&#115;&#116;&#32;&#104;&#97;&#118;&#101;&#32;&#55;&#32;&#100;&#97;&#121;&#115;&#32;&#116;&#111;&#32;&#115;&#101;&#110;&#100;&#32;&#117;&#115;&#32;&#116;&#104;&#101;&#32;&#66;&#105;&#116;&#67;&#111;&#105;&#110;&#32;&#97;&#102;&#116;&#101;&#114;&#32;&#55;&#32;&#100;&#97;&#121;&#115;&#32;&#119;&#101;&#32;&#119;&#105;&#108;&#108;&#32;&#114;&#101;&#109;&#111;&#118;&#101;&#32;&#121;&#111;&#117;&#114;&#32;&#112;&#114;&#105;&#118;&#97;&#116;&#101;&#32;&#107;&#101;&#121;&#115;&#32;&#97;&#110;&#100;&#32;&#105;&#116;&#39;&#115;&#32;&#105;&#109;&#112;&#111;&#115;&#115;&#105;&#98;&#108;&#101;&#32;&#116;&#111;&#32;&#114;&#101;&#99;&#111;&#118;&#101;&#114;&#32;&#121;&#111;&#117;&#114;&#32;&#102;&#105;&#108;&#101;&#115;</pre></html>EAAAAL64zV92G5v/huezRw/wRoeWSbwvGFHd5BmNXImLQMQueyWVUB4aGj3Z5l29zggjaSi5Snwc5quVNu1nbubNCqksA+XzDE+T9+ukEqbwCPErRLDdjc68URMkoWRhF/+DglEoUhJFSZb7Yv3zi8l5ybjwIS8vqPAfAplWftRZlEu1gZqHsJCoJW6icQzYthCz/bF5elu2b/2C43FKr4xkujuU1ADy+qUHNBxik5xtyz91h6A5mHHLdFjSnzRJQrxuGEujeBiLtuIdLEO2YiaUB1k=EAAAAEXWRR9Iido+6WIcgJ0neEp6YlC1UI0cHZFdVIYgj2IYWinDir.Properties.Resources”

Inspection: LoadModule, MemoryStream, ToBase64String, FileAccess, RSACryptoServiceProvider

***Publickey.keyxml:***

SHA256: e5f6ad503c88055b931c7af7ec52dfd09759330633b2ace4dba9722efd5c876

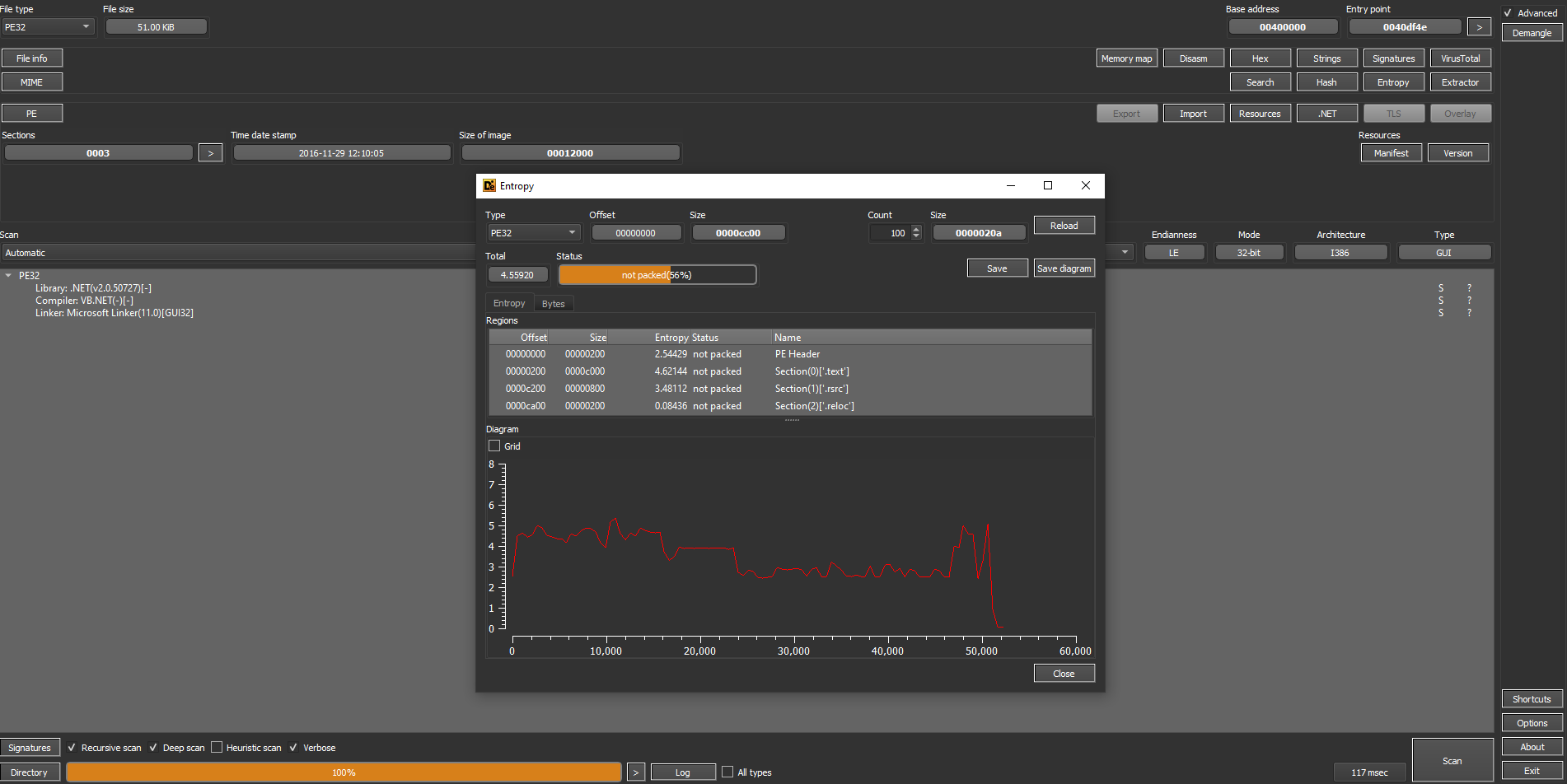
MD5: 4e3e18e6140c64cec89f4be5af25751

Interesting Strings: <RSAKeyValue><Modulus>5lRbjTmdM2okFAzONhfepTt7gTNMTsSTsXfbbc+ZjpCYjeRzFxD9+Qfxu+moaExNFJGwkwsTLzzX+36/Vszg85jhKImeTvHyLX2b5SnL93JGN9vchkuMEEcP4SNjzJWHvWxYuJL7vBj4sjV81Fxh4HsZsohED0FtMjAR2RP40YaNs/tfGlmFrPBAmeKJF4+uVuxkr0cHtdhhCH+/BPfdqobfMNQ1eIjZj+2IpVYoFPmSOeJFiCvLNEHm8McuEE9BOvSnIs5D8hqDbq0TZdoFuGLRpwaEPYqVNvAYh1H9z+kXkiA/TU/GedKJ5G7bffOkawta7vC7B5kb0DZLpNQw8Q==</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>

## Packing

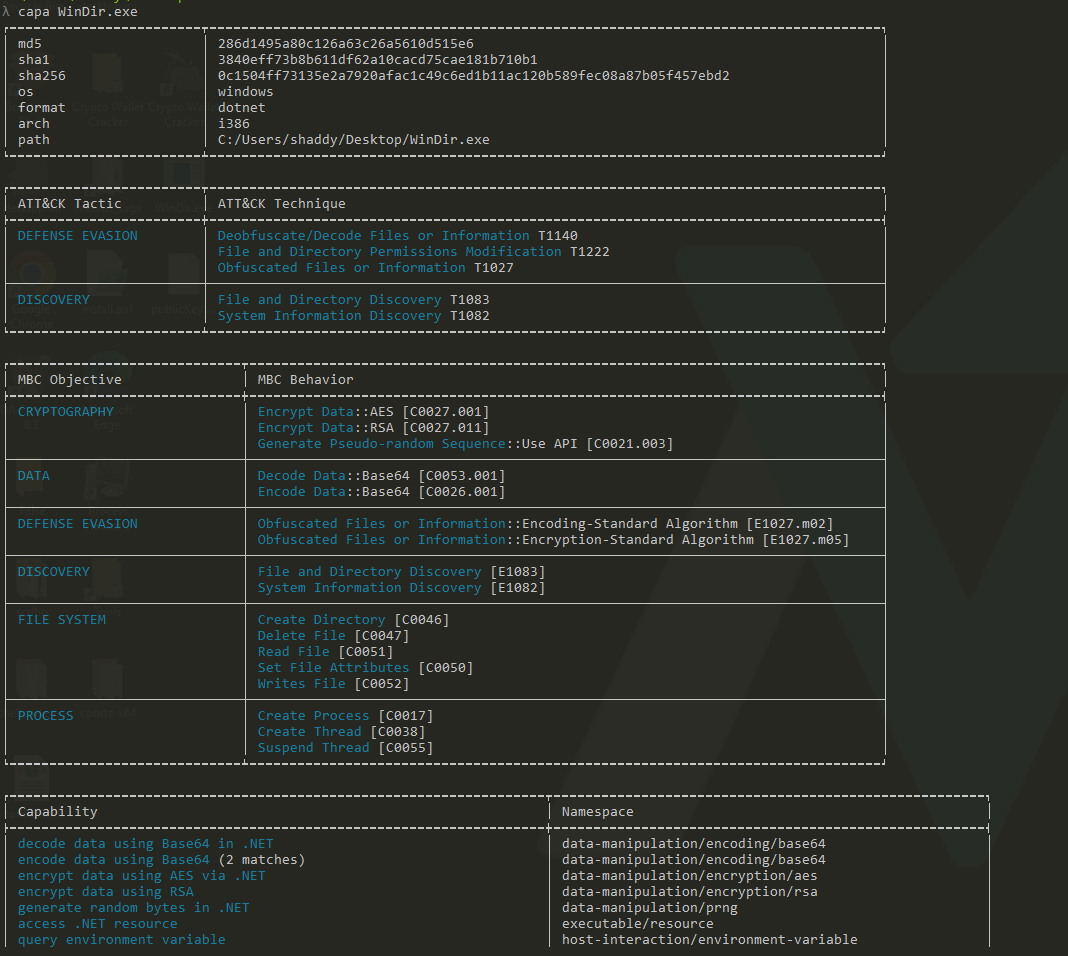
### Detect-It-Easy

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.



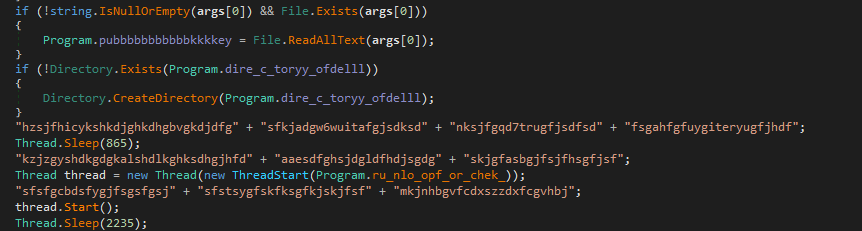
### Capa-Output

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.

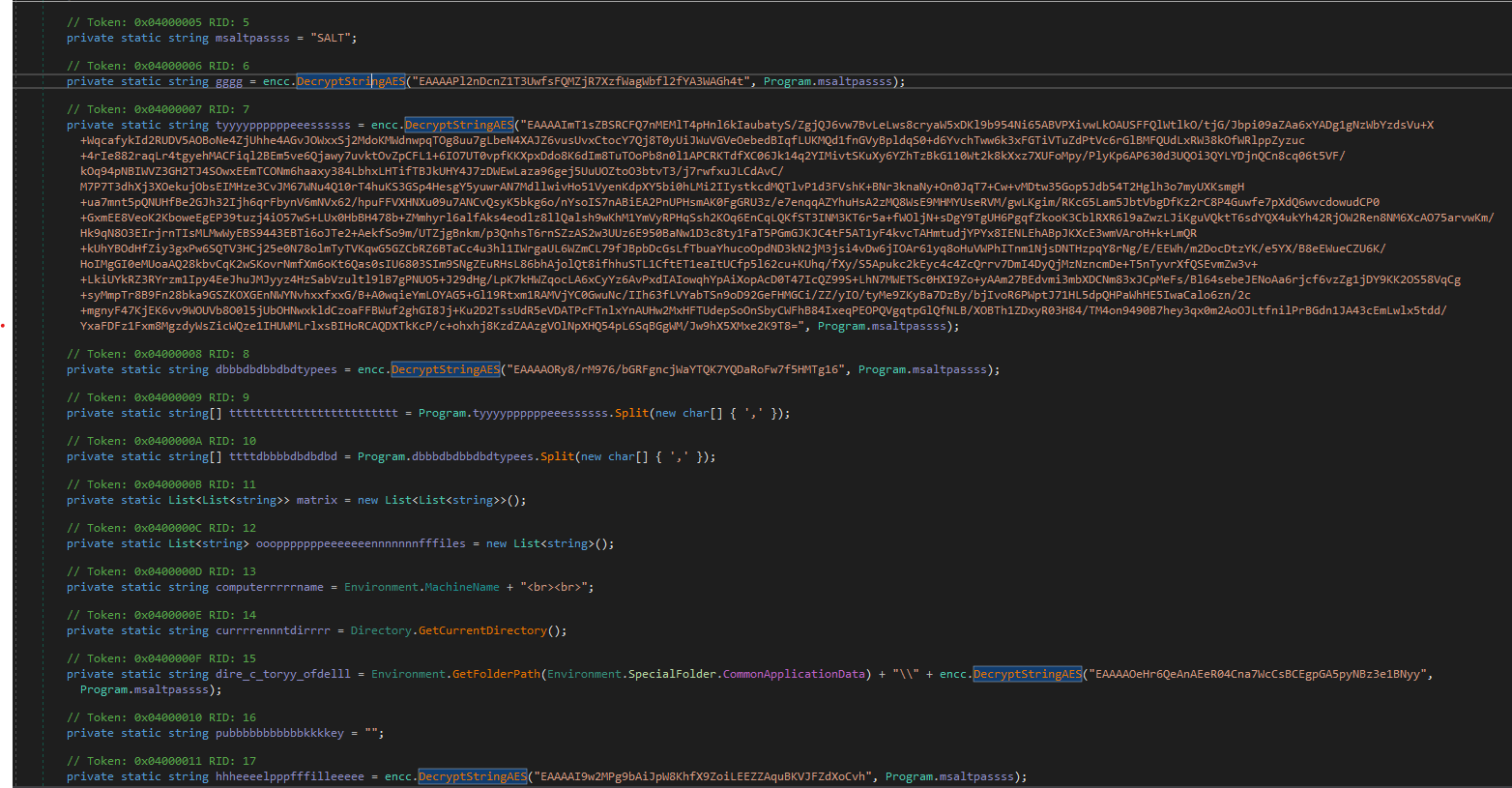


### Static Analysis

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.



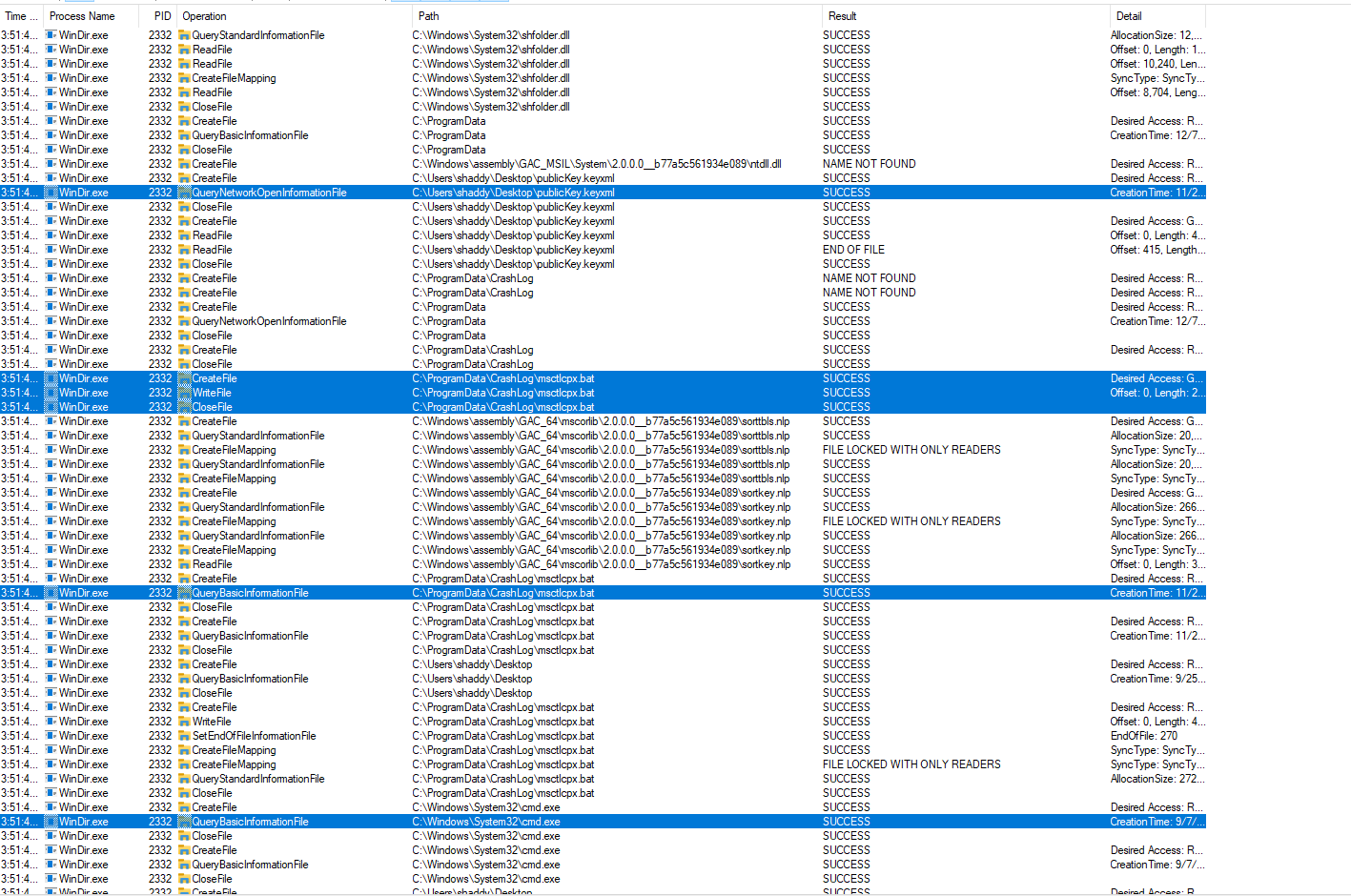
In my static analysis of SamSam ransomware, I found some encrypted bytes and string type variable which are storing the string value after decrypting the bytes. The SamSam ransomware is using the DecryptStringAES() function which were taking two parameter one the cipher text and the second was shared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



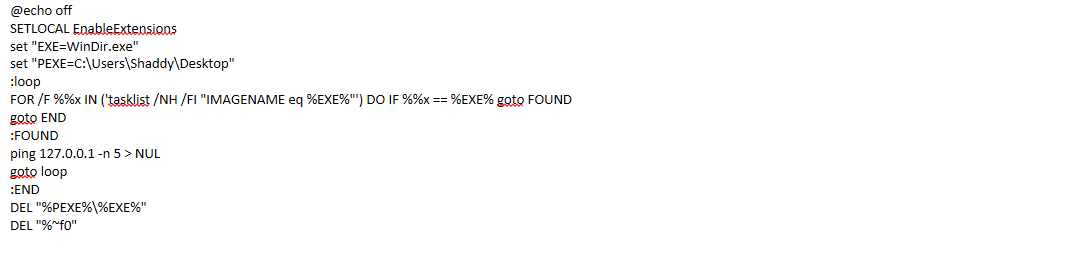
## Basic Dynamic Analysis

### Procmon and Process Hacker

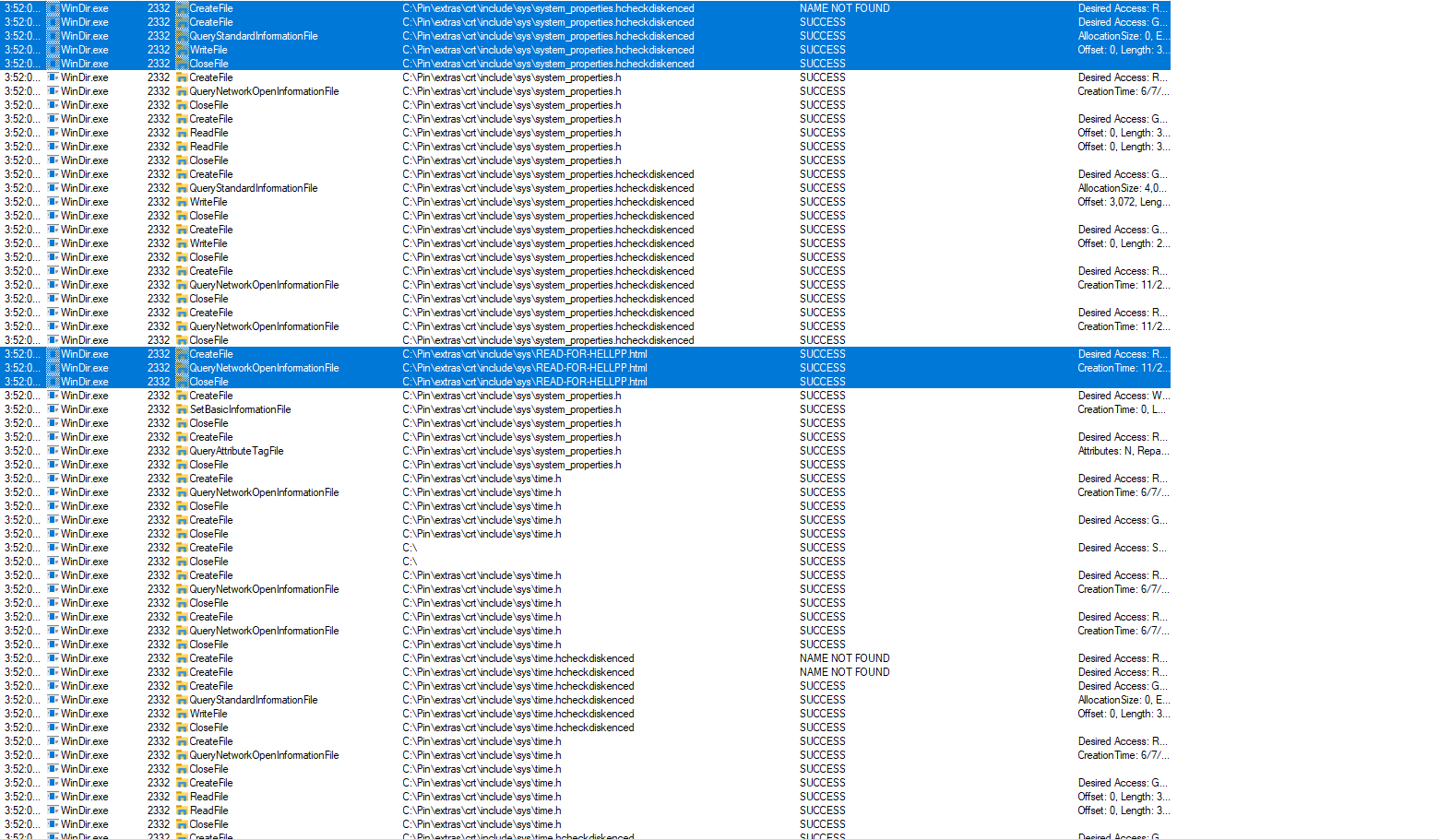
As an offensive security researcher, I always prefer Procmon and process hacker in my first detonation of malware sample which I analyze. When I executed the sample and captured all traffic using Wireshark and captured the all activities using Procmon, I noticed some interested activities on Procmon. 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.



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.

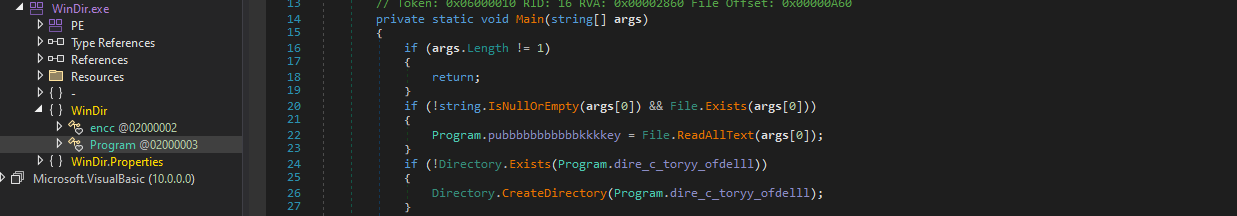


After some time, I found that my entire system was encrypted by SamSam Ransomware. So, I checked the encrypted file it was adding the extension (checkdiskenced) and leaving the ransom note with the name of READ-FOR-HELLPP.html.



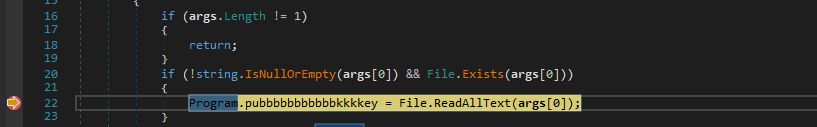
## Advanced Dynamic Analysis

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.

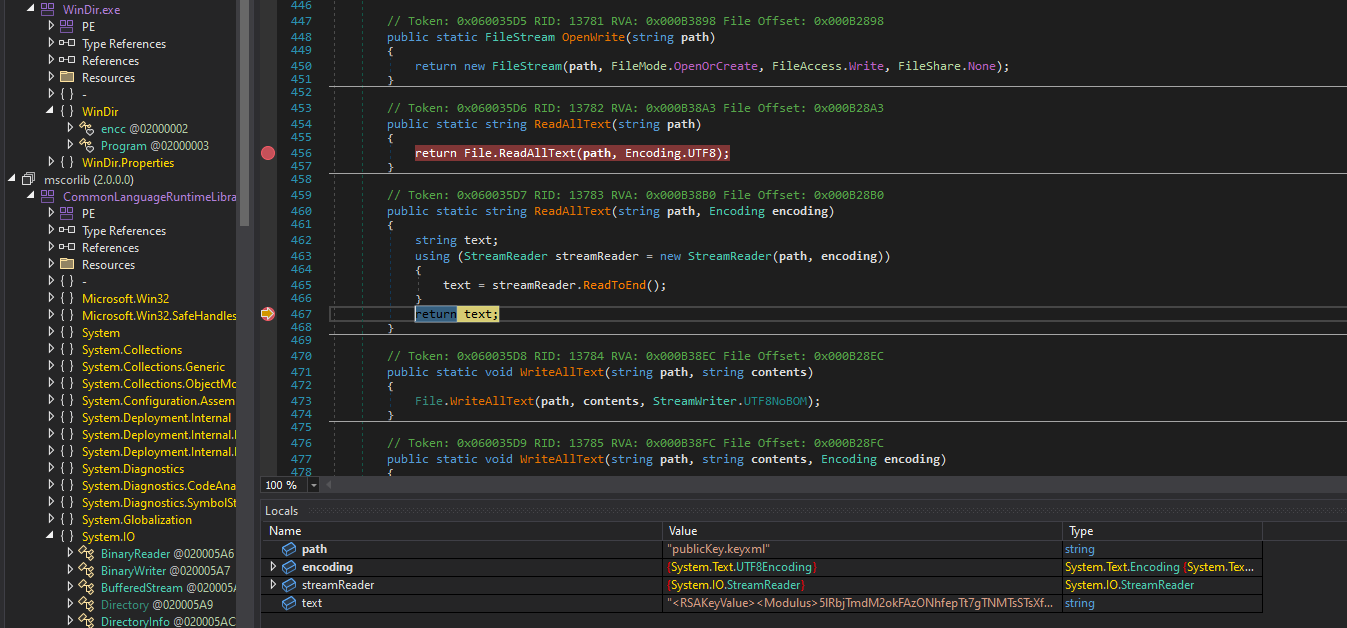


### Breakpoints:

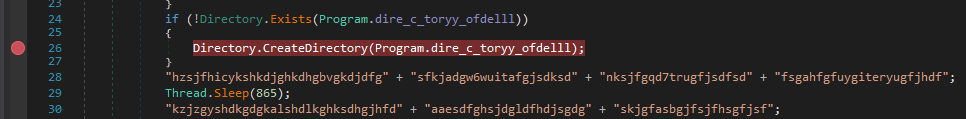
I start analysis step by step and put breakpoints. First, I wanted to know about the file it was taking as an argument then the bytes it was storing in the variable after reading the argument file.



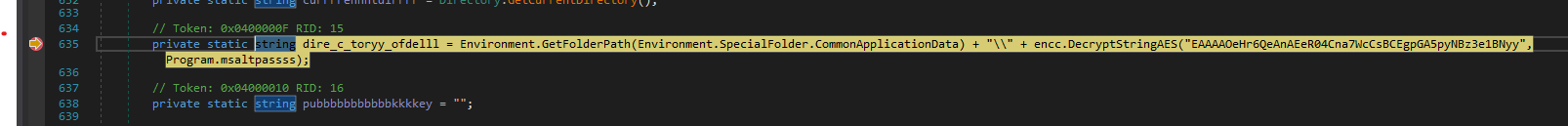
Following the execution flow and putting breakpoints, I found that the Samsam ransomware was looking “publicKey.keyxml” file and reading all bytes which indicates the public key of RSA-2048 encryption algorithm that was using for encryption in this sample.



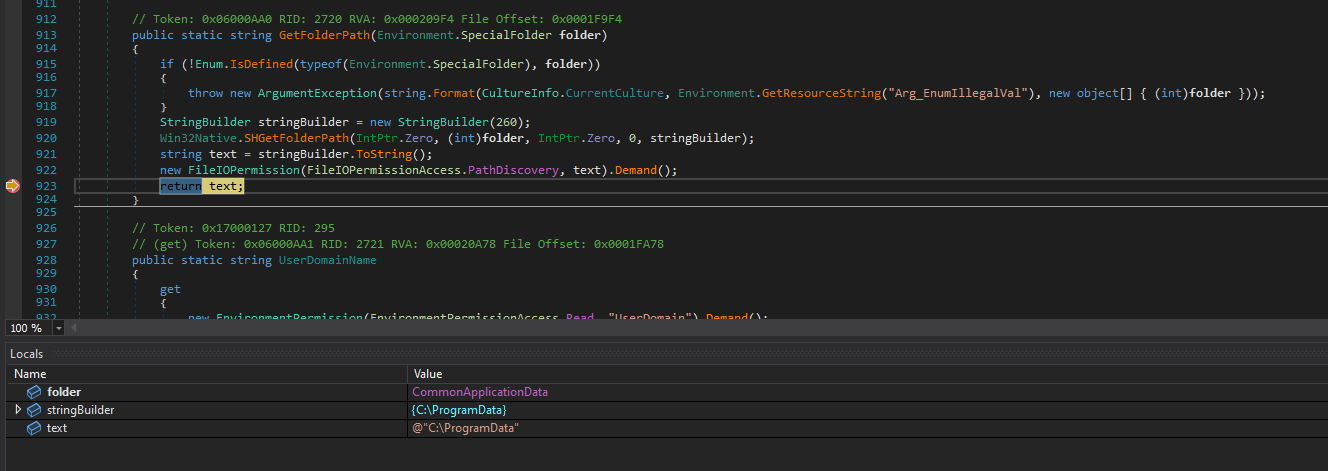
After successfully taking the argument, the malware was looking some directory, if the directory already exists than it was creating a new thread and executing something in new thread. But at that time for me important thing was to know about the directory it was looking, so I put breakpoints on following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



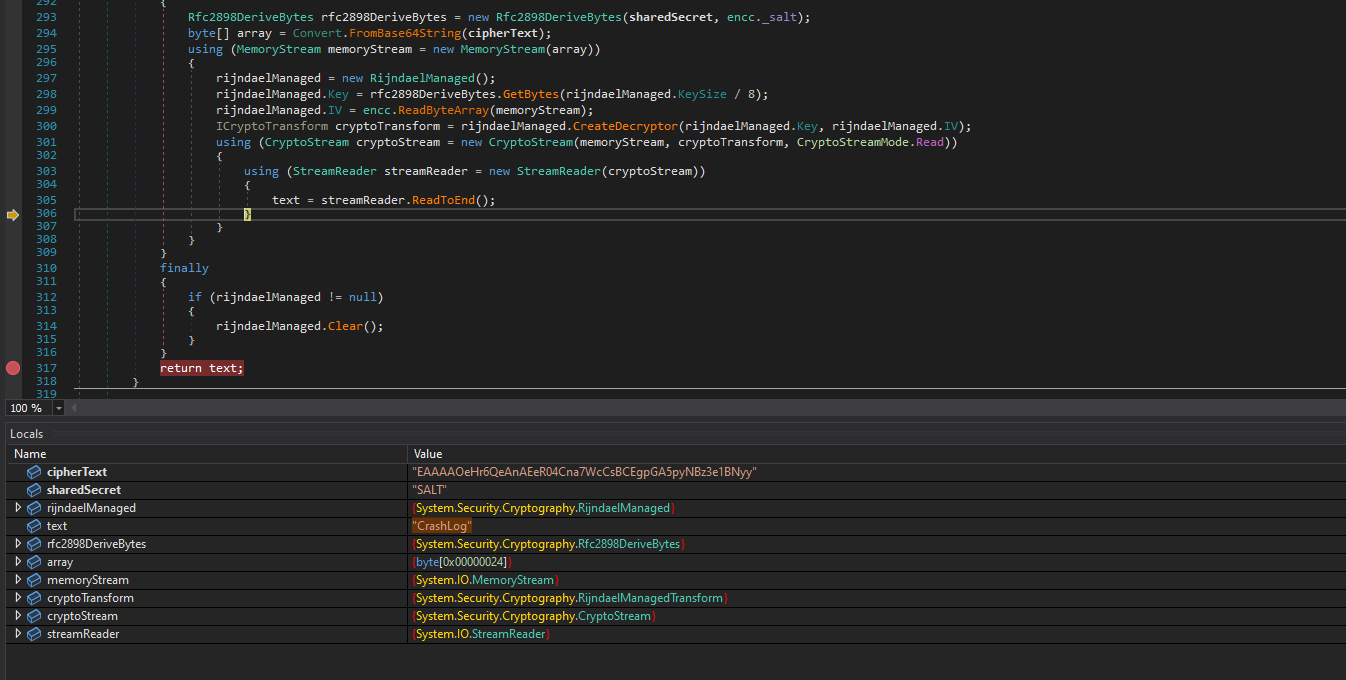
When I follow the execution, I found that it was creating a directory, the path and name of directory was encrypted with AES encryption.



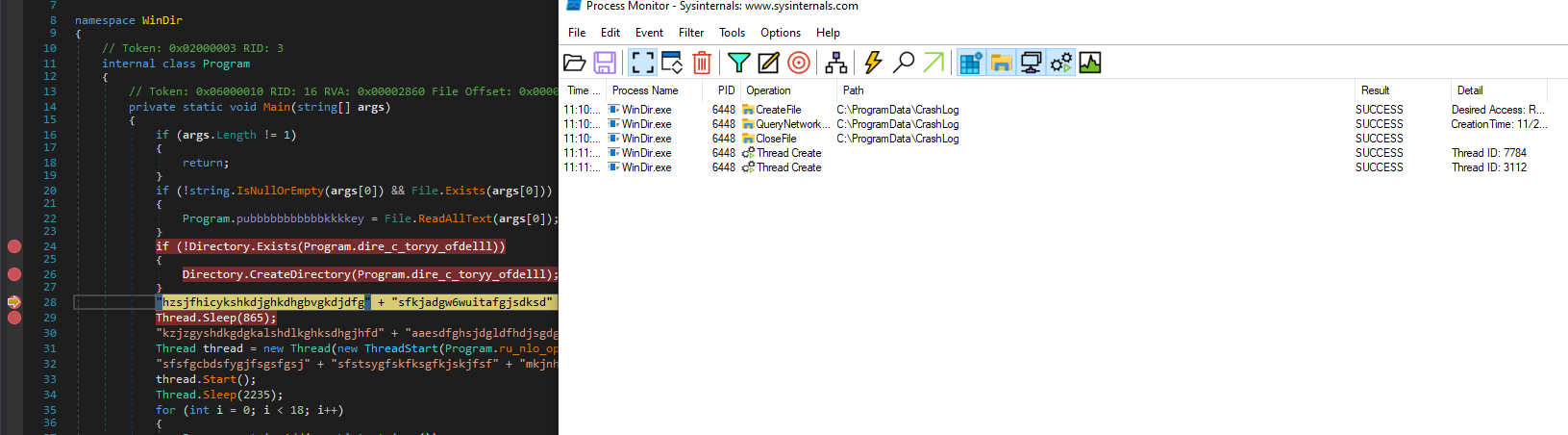
After debugging, I found the decrypted directory path. Samsam was looking the directory “C:\ProgramData\CrashLog”.



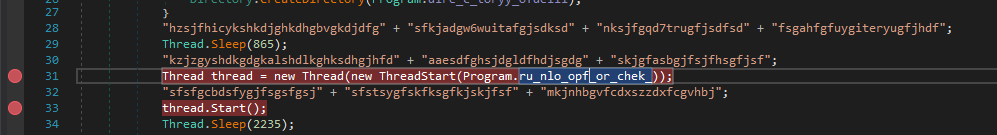
When I decrypted the name of subdirectory it was appending with path of folder “C:\ProgramData\”. Samsam sample was looking for the path of that directory. “C:\ProgramData\CrashLog\”.



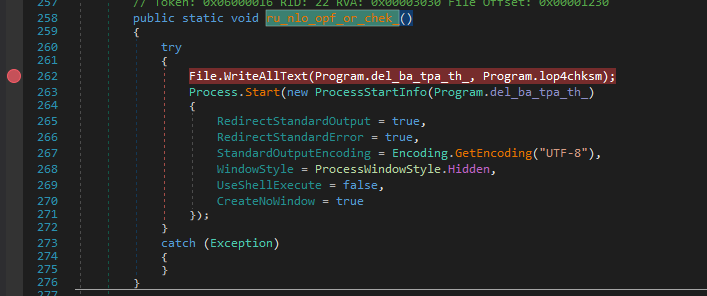
For the verification of created directory, I just executed the instructions of directory creation function and recorded all activities using Procmon. I found on the Procmon that same results that I extracted from debugging and putting breakpoints.



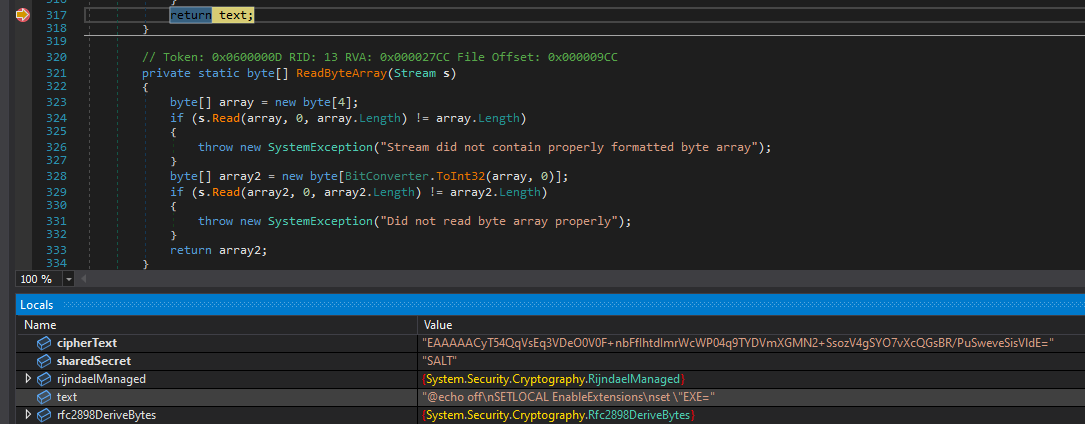
After creating the directory, the instructions were to create a new thread and start. Samsam ransomware was starting something as a new thread. Now I wanted to know about the process it was starting as a new thread. Before creating the new thread there was some junk strings and random sleeps which at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



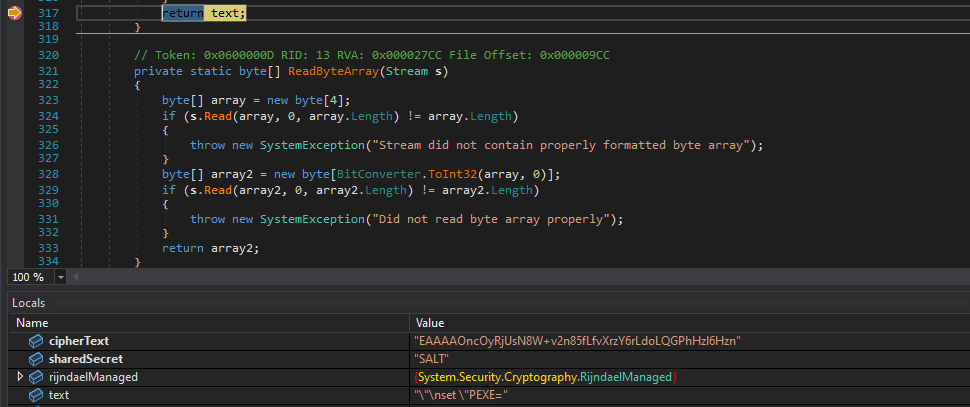
When I followed the execution follow, I found that it was starting a function **“rul\_nlo\_opf\_or\_chek\_”** as a thread and in that function, there was new things were happening. This function was writing some content in file at that point, I don’t know about the file name, the path where the file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



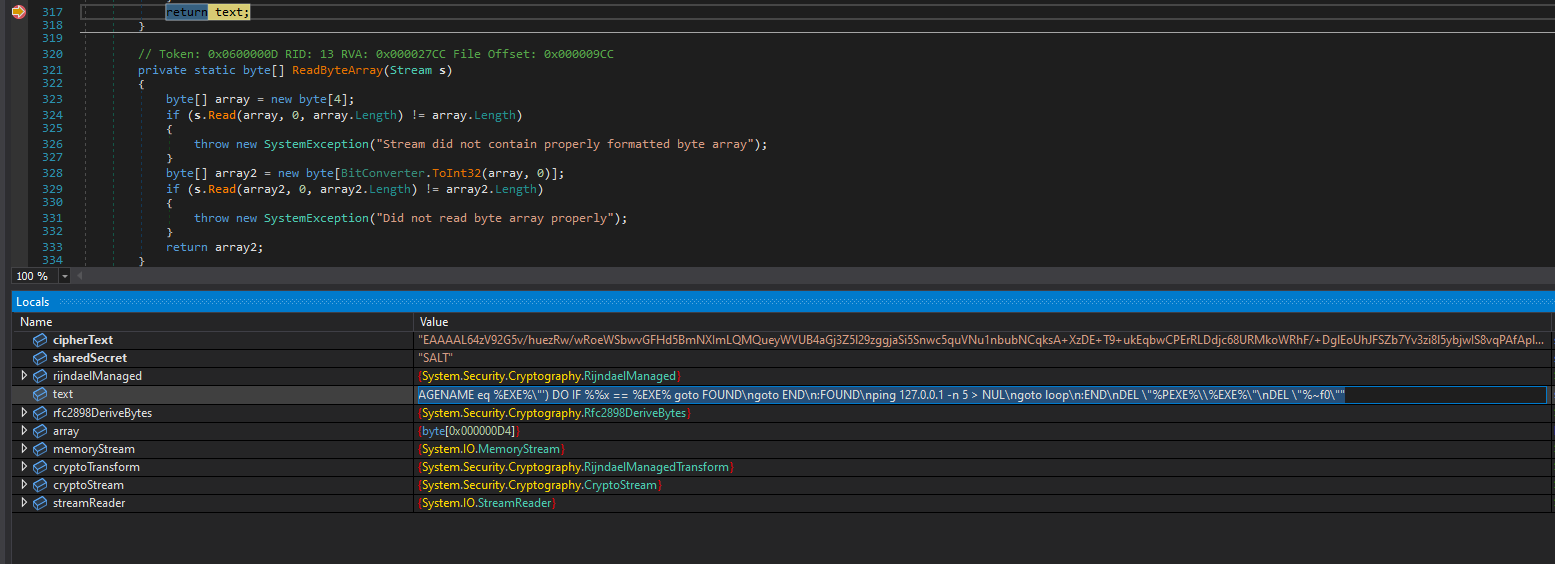
So, at that point my main target was to know about the file path and the content written in that file. I started my analysis in the same flow and put the breakpoints on each function return statement and check the content and path of that file. I found that the name of file and the content was fully 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.



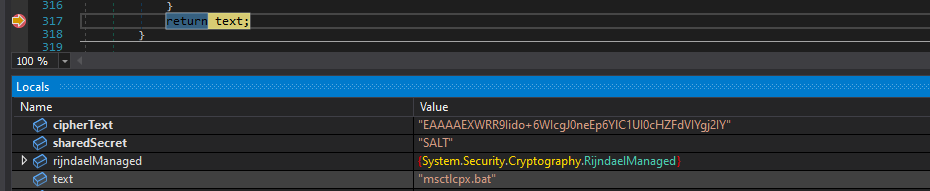
The second string after decrypting was **"\"\nset \"PEXE=".** It looks that it was setting or assigning the PE file to a variable.



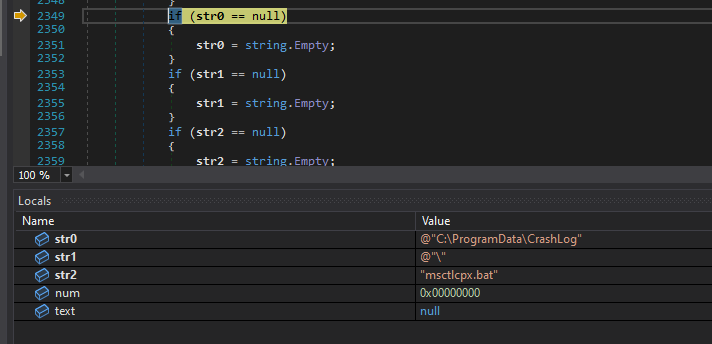
The last string that I found was **"\"\n:loop\nFOR /F %%x IN ('tasklist /NH /FI \"IMAGENAME eq %EXE%\"') DO IF %%x == %EXE% goto FOUND\ngoto END\n:FOUND\nping 127.0.0.1 -n 5 > NUL\ngoto loop\n:END\nDEL \"%PEXE%\\%EXE%\"\nDEL \"%~f0\"".** This looks a loop which is looking some specific process in running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



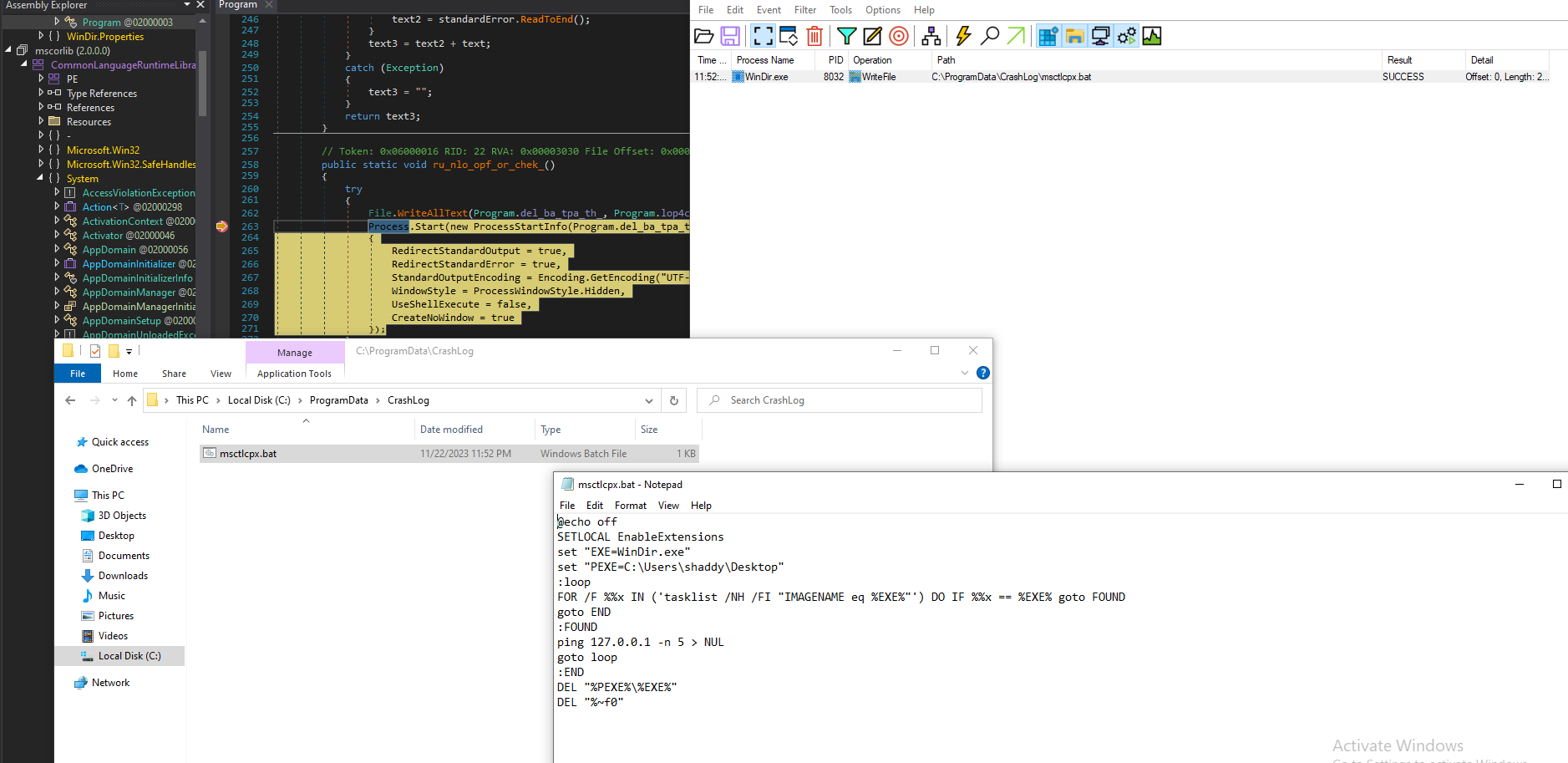
At that point, I found all the content which were written in some file and was executing but still I was not aware of the file name and the path where it was written. So, I decrypted the file name that was a batch file with the name of **“msctlcpx.bat”.**



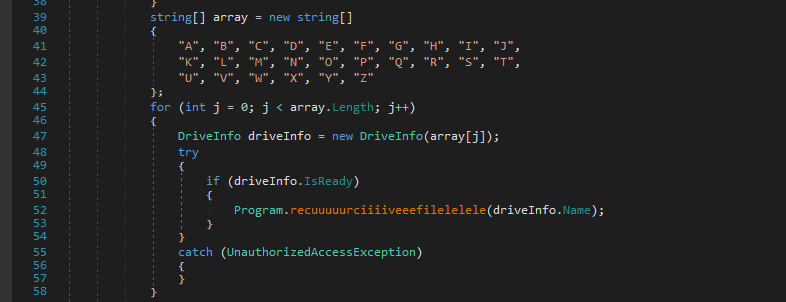
Now for this execution flow, I final step was to know about the path where this batch file was written. So, I decrypted the path of that file. The batch file was written in the same directory.



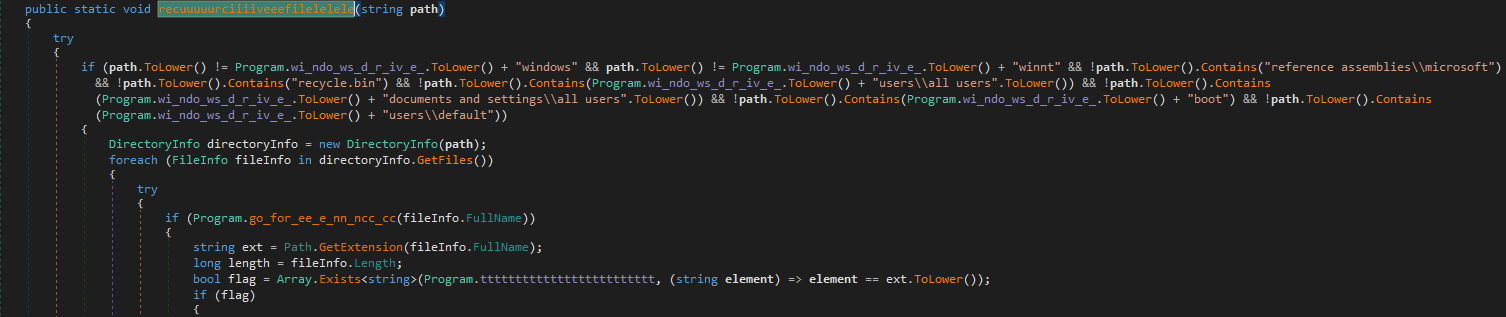
Now for the verification of the whole content, I captured all activities using Procmon and executed only the instruction that were writing batch file into above mentioned directory. So I found the batch file with the same name and content.



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.

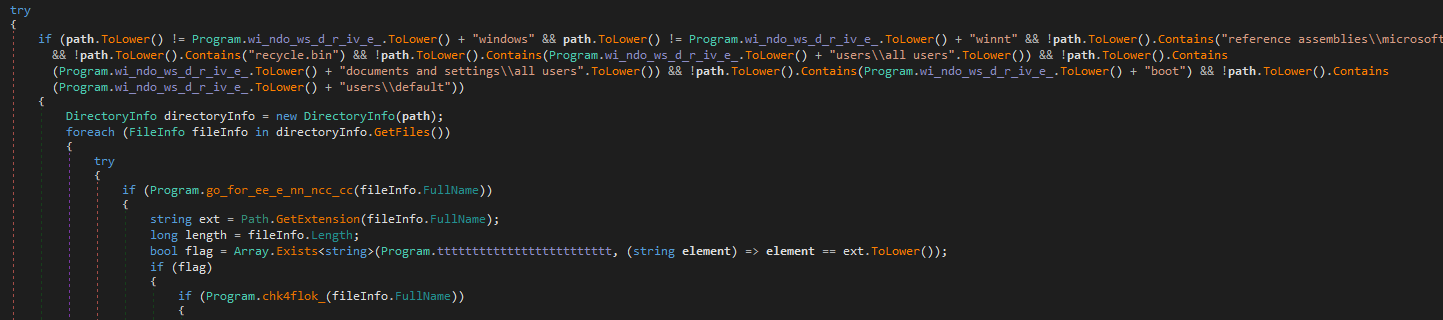


After exploring the function there were conditions seem to be related to filtering out specific directories or paths. Here's a breakdown of what each part of the condition is checking:

* 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\_.
* 

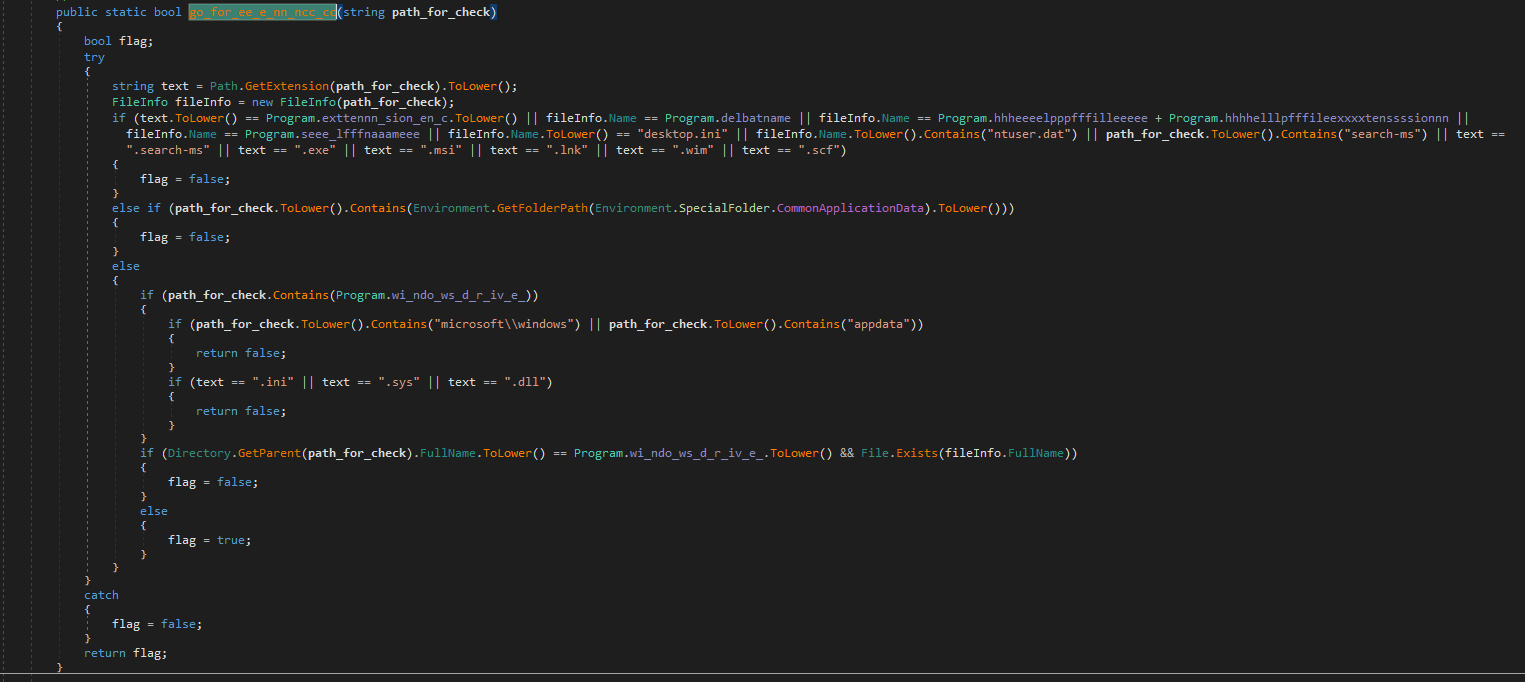
this conditional statement is likely used to filter out specific paths or directories based on the mentioned conditions. If the given path meets any of these conditions, the overall condition would evaluate to true, indicating that the path should be excluded or not processed further.

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.

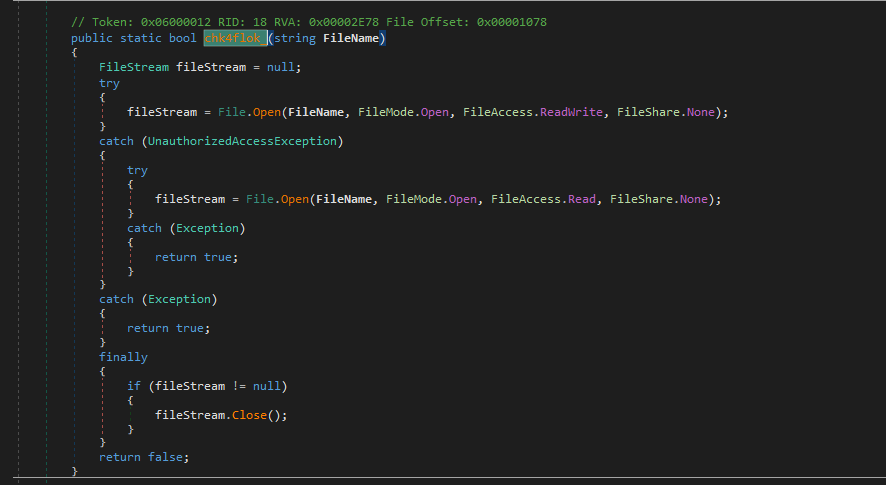
* text.ToLower() == Program.exttennn\_sion\_en\_c.ToLower(): Checks if the lowercase version of the given text is equal to the lowercase version of Program.exttennn\_sion\_en\_c.
* fileInfo.Name == Program.delbatname: Checks if the name of the file (retrieved from fileInfo) is equal to Program.delbatname.
* 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").



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.ttttttttttttttttttttttttt, (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.



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.

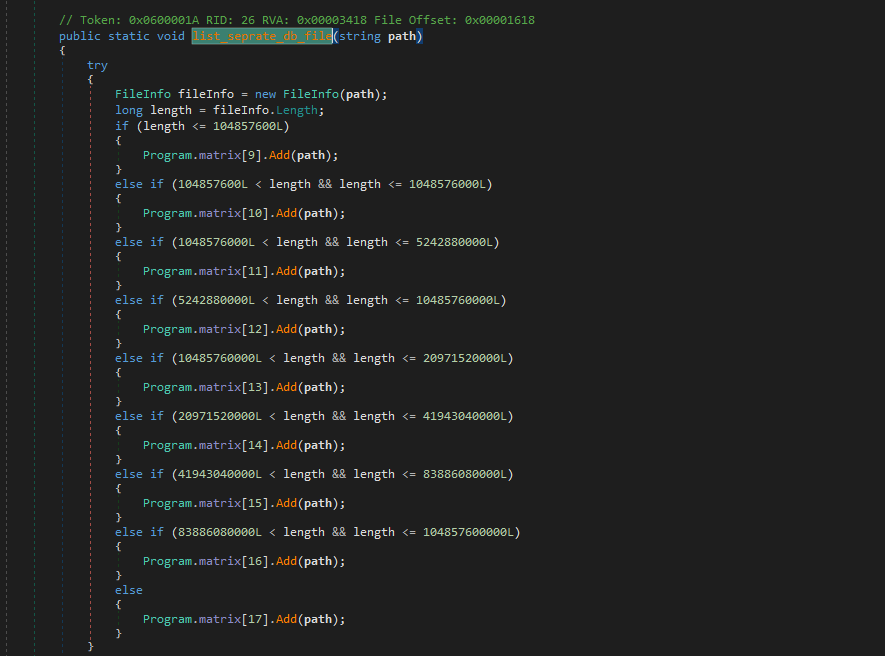
***bool flag2 = Array.Exists<string>(Program.ttttdbbbbdbdbdbd, (string element) => element == ext.ToLower());***

***if (flag2)***

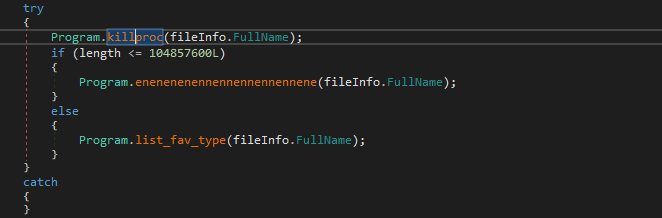
***{***

***Program.list\_seprate\_db\_file(fileInfo.FullName);***

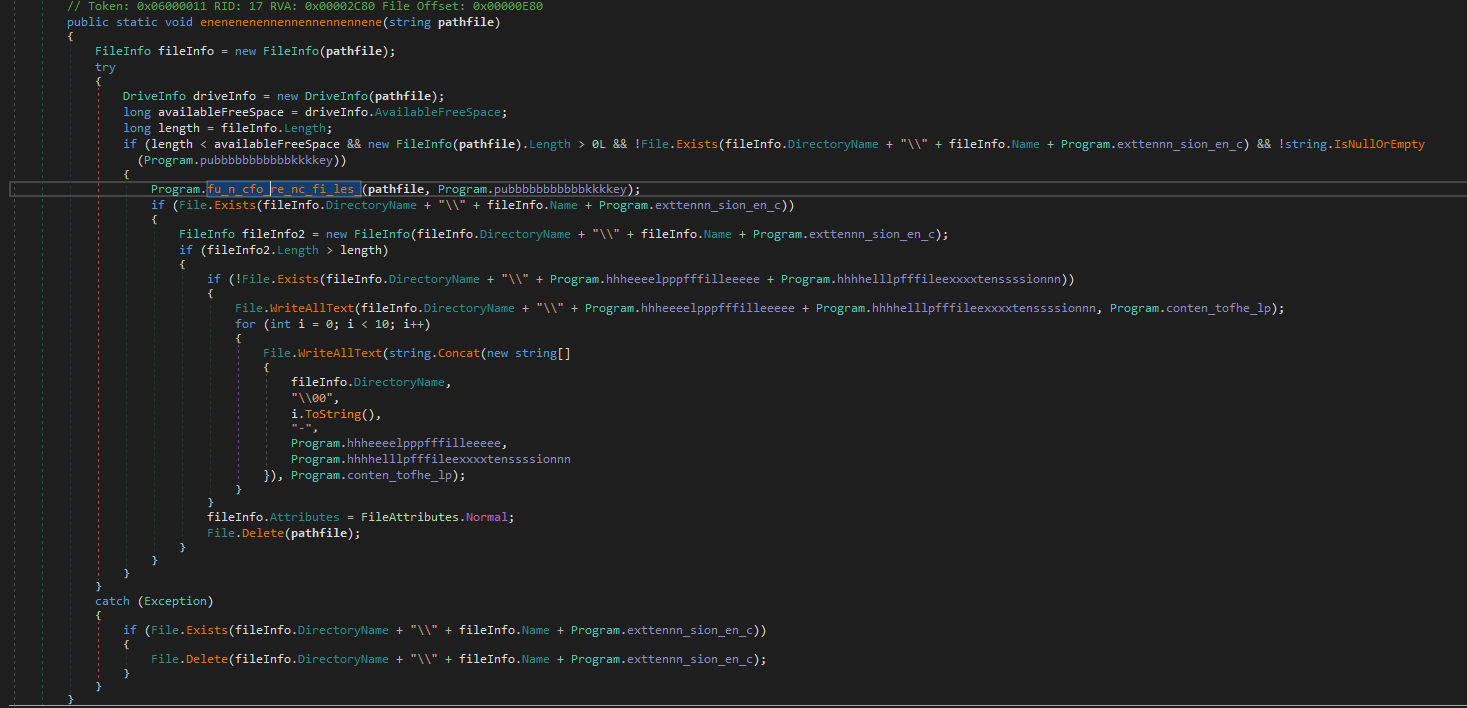
***}***

******

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.



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.

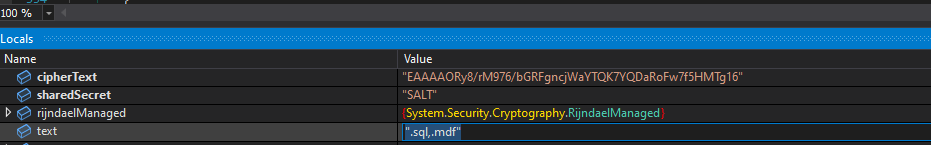


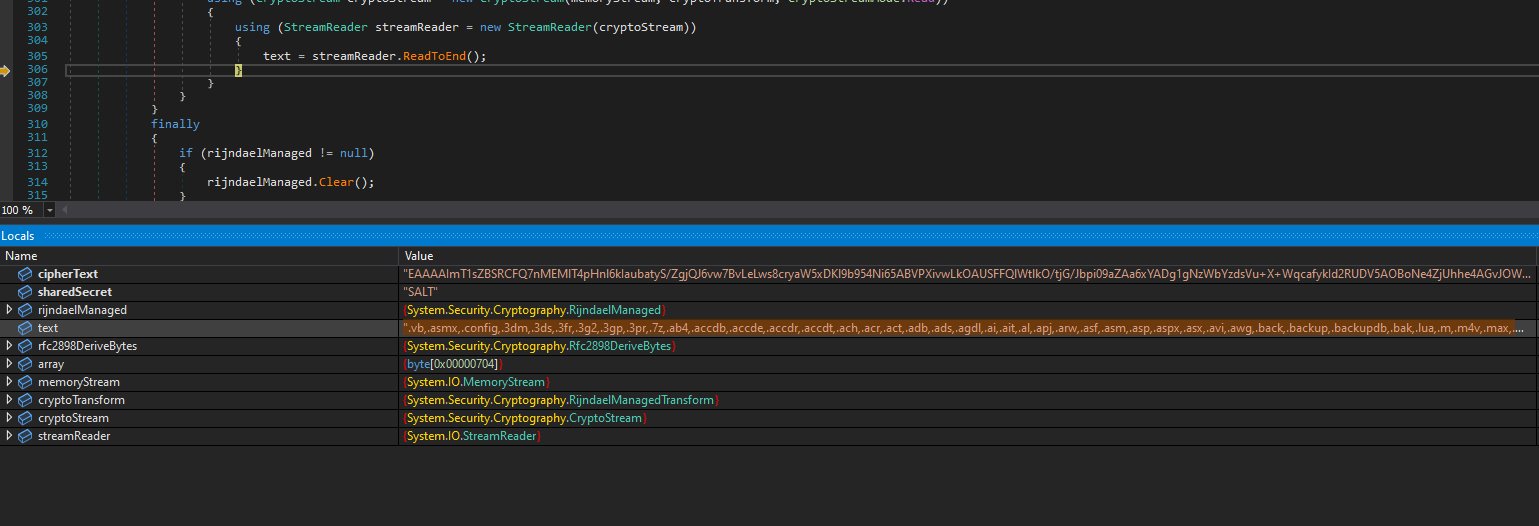
Now before concluding the analysis, I want to show you guys the extensions it was looking for and the database file types.

".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,.dxg,.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"

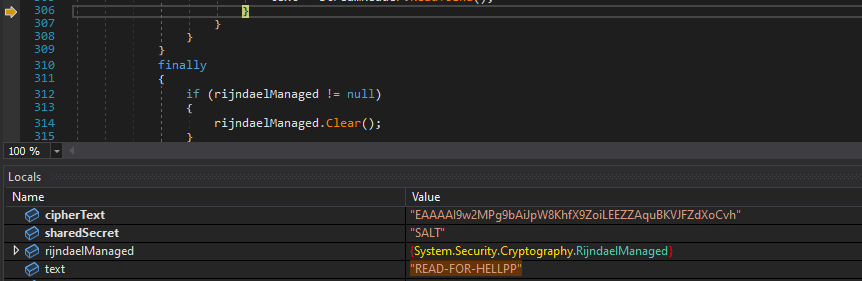
***Database Files***

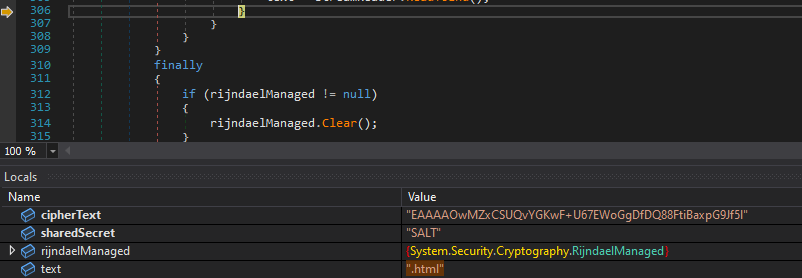
".sql,.mdf"





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.

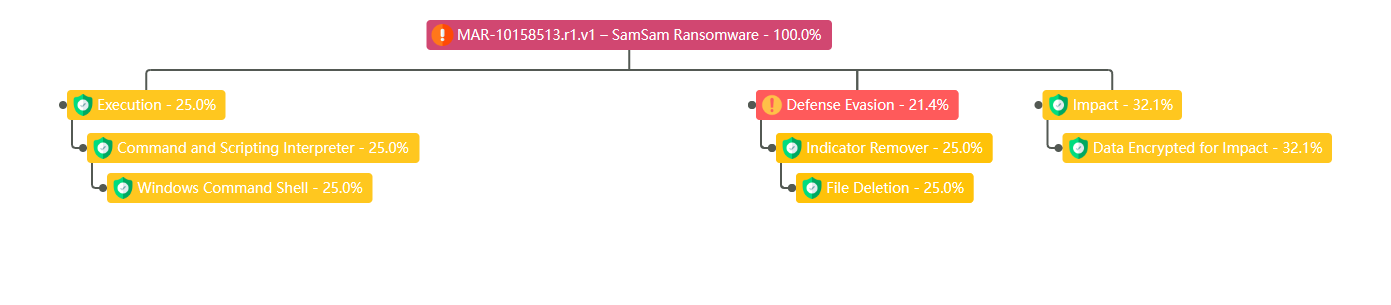
# Extracted TTP’s

## MITRE ATT&CK MAPPING

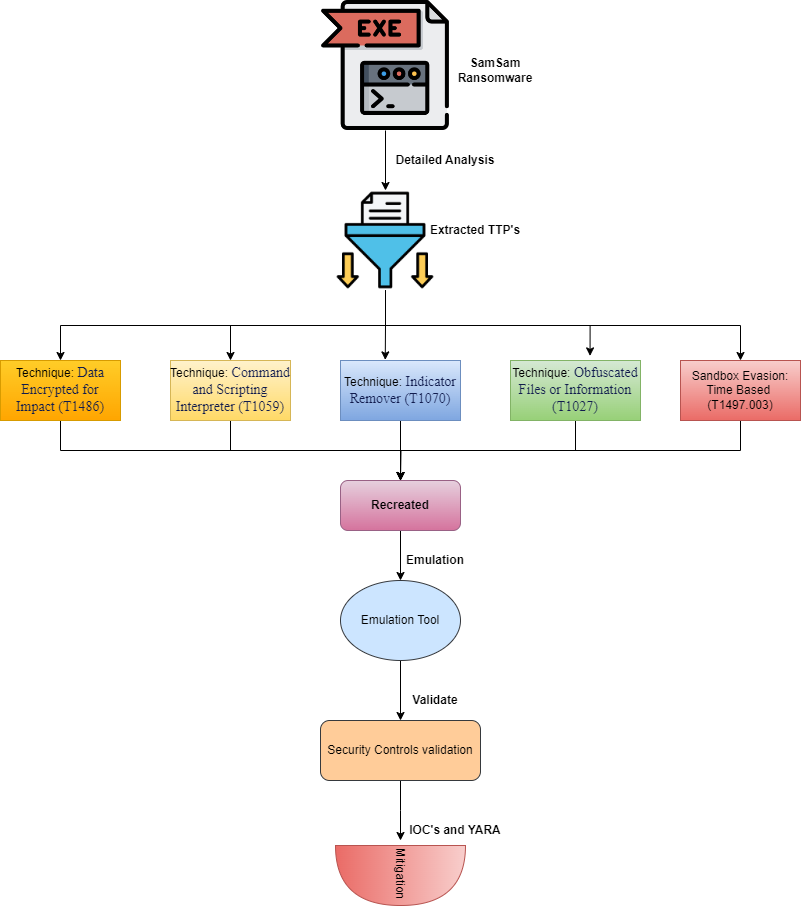
|  |  |
| --- | --- |
| Tactic | Techniques and Sub-Techniques |
| Defense Evasion | Technique: Time-Based Sandbox Evasion |
| Impact | Technique: Data Encrypted for Impact (T1486) |
| Execution | Technique: Command and Scripting Interpreter (T1059)  Sub\_technique: Windows Command Shell (S003) |
| Defense 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>xPN1oBWSqfQgInnB6ydF204jiHN/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.