


# An Inside Look at How Ryuk Evolved Its Encryption and Evasion Techniques

 [labs.sentinelone.com/an-inside-look-at-how-ryuk-evolved-its-encryption-and-evasion-techniques/](https://labs.sentinelone.com/an-inside-look-at-how-ryuk-evolved-its-encryption-and-evasion-techniques/)

Marco Figueroa



## Introduction

In the last three months, there has been a 50% uptick in ransomware, with the Ryuk ransomware garnering the most attention after a string of high profile attacks that have been crippling companies. Last month it was reported that Ryuk hit UHS hospital networks with force, spreading across UHS healthcare facilities in the US from coast to coast. This well-orchestrated attack left many hospital workers without access to labs, radiology, and patient records, which led to workers having to resort to pen and paper to triage patients. Ryuk is currently attacking approximately 20 organizations a week, and this number will only expand due to its successes.

There are a number of factors that contribute to this success. These include its harnessing of other “toolkits” such as TrickBot and Emotet, and being quick to jump on newly-exposed vulnerabilities such as Zerologon. We also see that Ryuk has iterated since its earlier incarnations to evade detection and markedly improve the time it takes from execution to full encryption, making life increasingly difficult for organizations that cannot respond to threats at such speed.

In this post, we look at how Ryuk has evolved since 2018 and explore the improvements in encryption speed and evasion techniques that we see in Ryuk samples today. Along the way, we detail a method analysts can use to extract the Ryuk executable from memory and dump it to file for further inspection.

## Ryuk Overview

---

When Ryuk ransomware burst onto the scene, it was initially believed that it was developed by the same threat actors who developed Hermes Ransomware. However, it was later discovered that Hermes was being sold on the black market, allowing cybercriminals to purchase the framework and convert it to what is known today as Ryuk.

The current waves of attacks have been known to use a combination of Emotet, Trickbot, and Ryuk. In recent weeks, the actors behind Ryuk have even been observed using ZeroLogon to extend their reach and broaden the delivery of their ransomware payloads. While the Ryuk payloads do not specifically contain the ZeroLogon functionality, the flaw is being leveraged at earlier stages in the attack chain. Attackers are able to use existing capabilities in Cobalt Strike and similar frameworks to achieve the privilege escalation. It is quickly becoming clear that ZeroLogon will become a staple in the attackers' collective "toolbelt".

## Reversing and Comparing Ryuk 2018 and 2020

---

There are many tools in the reversing ecosystem for diffing binaries like Bindiff, but a fast tool when comparing binaries I find most useful is Ghidra's version tracking to check for comparisons between binary files.

If we compare an earlier version of Ryuk with the latest version, we can note some interesting changes. In the most recent version, Ryuk obfuscates its hardcoded strings to become more difficult for AV vendors to detect:

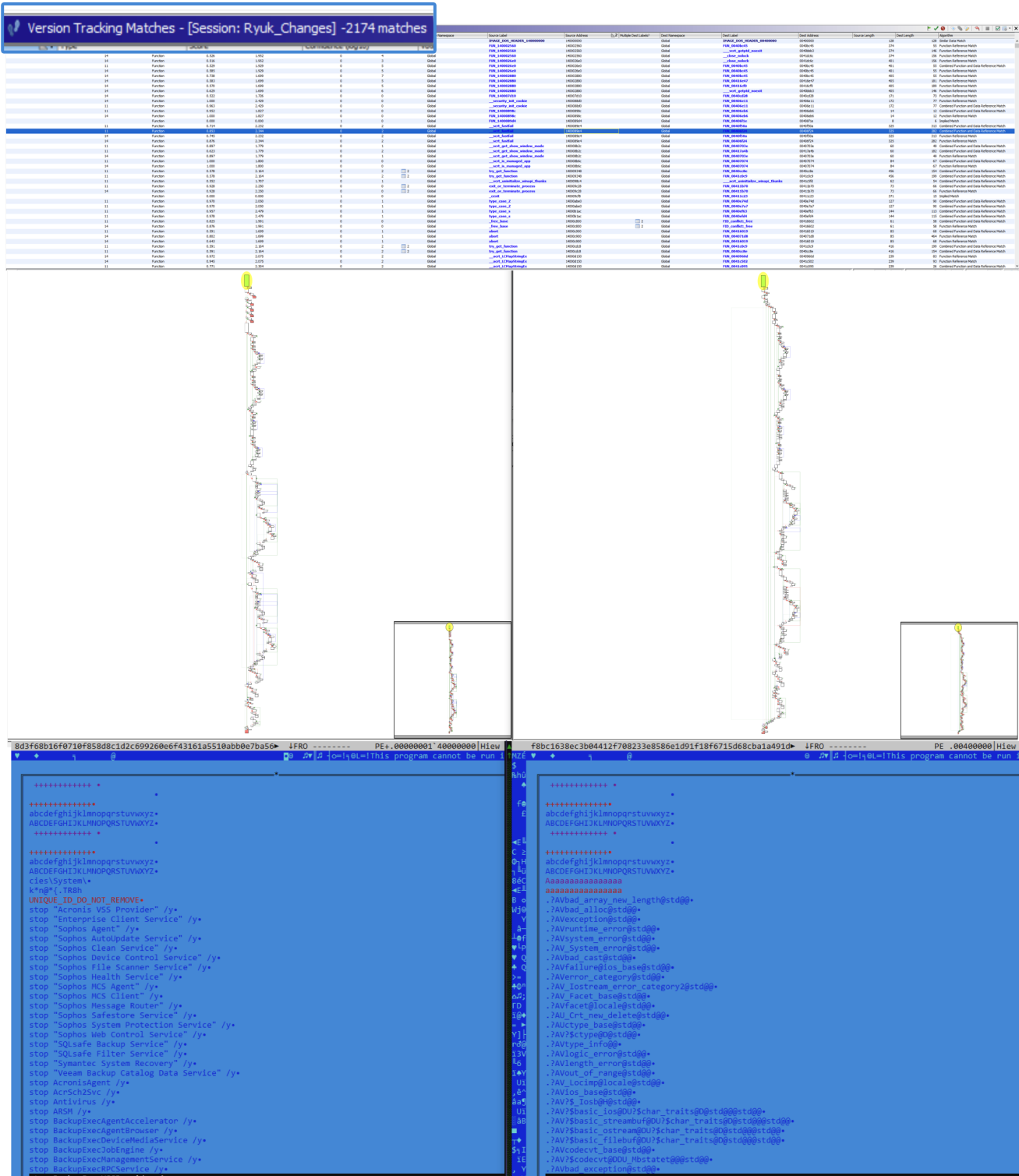


Figure 1: Ryuk 2018 vs 2020

Ryuk 2020 also copies itself to increase the speed of encryption, which we discuss in detail below.

The ransomware uses RSA and AES to encrypt files with extension `.ryk`, creating a new thread for each file it encrypts. Ryuk also uses the `CryptGenRandom` API, which fills the buffer with random bytes to generate a data encryption key.

```

74823d90 brcrypt!BCryptOpenAlgorithmProvider (no parameter info)
74824270 brcrypt!BCryptCloseAlgorithmProvider (no parameter info)
748242f0 brcrypt!BCryptEnumAlgorithms (no parameter info)
74824600 brcrypt!BCryptEnumProviders (no parameter info)
74824a30 brcrypt!BCryptFreeBuffer (no parameter info)
74824a80 brcrypt!BCryptGetProperty (no parameter info)
74824d40 brcrypt!BCryptSetProperty (no parameter info)
74824e00 brcrypt!BCryptGenerateSymmetricKey (no parameter info)
74825060 brcrypt!BCryptGenerateKeyPair (no parameter info)
748251e0 brcrypt!BCryptFinalizeKeyPair (no parameter info)
748252a0 brcrypt!BCryptEncrypt (no parameter info)
74825500 brcrypt!BCryptDecrypt (no parameter info)
74825830 brcrypt!BCryptExportKey (no parameter info)
74825ce0 brcrypt!BCryptImportKey (no parameter info)
74825f00 brcrypt!BCryptImportKeyPair (no parameter info)
748264d0 brcrypt!BCryptSecretAgreement (no parameter info)
74826610 brcrypt!BCryptDeriveKey (no parameter info)
74826760 brcrypt!BCryptDuplicateKey (no parameter info)
74826920 brcrypt!BCryptDestroyKey (no parameter info)
74826a50 brcrypt!BCryptDestroySecret (no parameter info)
74826b20 brcrypt!BCryptCreateHash (no parameter info)
74826f40 brcrypt!BCryptHashData (no parameter info)
74827060 brcrypt!BCryptInitHash (no parameter info)
74827170 brcrypt!BCryptCreateMultiHash (no parameter info)
74827310 brcrypt!BCryptProcessMultiOperations (no parameter info)
74827390 brcrypt!BCryptDestroyHash (no parameter info)
74827400 brcrypt!BCryptDuplicateHash (no parameter info)
748276b0 brcrypt!BCryptSignHash (no parameter info)
74827970 brcrypt!BCryptVerifySignature (no parameter info)
74827c00 brcrypt!BCryptGenRandom (no parameter info)
74827ca0 brcrypt!BCryptDeriveKeyCap (no parameter info)
748282f0 brcrypt!BCryptDeriveKeyPBDF2 (no parameter info)
74828600 brcrypt!BCryptKeyDerivation (no parameter info)
74828790 brcrypt!BCryptSetHashingInterface (no parameter info)
748287a0 brcrypt!BCryptHash (no parameter info)
74829510 brcrypt!BCryptGetLapsAlgorithmNode (no parameter info)
74829510 brcrypt!BCryptRegisterProvider (no parameter info)
74829d60 brcrypt!BCryptUnregisterProvider (no parameter info)
74829d70 brcrypt!BCryptQueryProviderRegistration (no parameter info)
74829d70 brcrypt!BCryptEnumRegisteredProviders (no parameter info)
7482a000 brcrypt!BCryptCreateContext (no parameter info)
7482db10 brcrypt!BCryptDeleteContext (no parameter info)
7482dc10 brcrypt!BCryptEnumContexts (no parameter info)
7482d000 brcrypt!BCryptConfigureContext (no parameter info)
7482d000 brcrypt!BCryptQueryContextConfiguration (no parameter info)
7482d090 brcrypt!BCryptAddContextFunction (no parameter info)
7482e0d0 brcrypt!BCryptRemoveContextFunction (no parameter info)
7482e200 brcrypt!BCryptEnumContextFunctions (no parameter info)
7482e300 brcrypt!BCryptConfigureContextFunction (no parameter info)
7482e4e0 brcrypt!BCryptQueryContextFunctionConfiguration (no parameter info)
7482e8b0 brcrypt!BCryptAddContextFunctionProvider (no parameter info)
7482e910 brcrypt!BCryptRemoveContextFunctionProvider (no parameter info)
7482e970 brcrypt!BCryptEnumContextFunctionProviders (no parameter info)
7482eb30 brcrypt!BCryptSetContextFunctionProperty (no parameter info)
7482ec00 brcrypt!BCryptQueryContextFunctionProperty (no parameter info)
7482d995 mov esi,dword ptr [brcrypt!BCryptResolveProviders+0x5408 (74834638)]
7482d998 ecx,esi
748270d0 call dword ptr [brcrypt!BCryptResolveProviders+0x5Fe8 (74835168)]
74827b03 call esi
74827ba9 mov eax,edi
74827ba9 lea esp,[ebp-20h]
74827baa pop edi
74827bab pop esi
74827bac pop ebx
74827bad mov ecx,dword ptr [ebp-4]
74827bb0 xor ecx,ebp
74827bb0 call brcrypt!BCryptResolveProviders+0x2f1c (7483209c)
74827bb3 leave
74827bb6 ret 1ch
74827bb8 int 3
74827bbc int 3
74827bbd int 3
74827bbe int 3
74827bbf int 3
brcrypt!BCryptGenRandom:
74827bc0 mov edi,edi
74827bc2 push ebp
74827bc3 mov ebp,esp
74827bc4 and esp,0FFFFFFFh
74827bc8 push ecx
74827bc9 push esi
74827bca mov eax,dword ptr [brcrypt!BCryptResolveProviders+0x4e00 (74834000)]
74827bcb mov esi,dword ptr [ebp+14h]
74827bcd cmp eax,offset brcrypt!BCryptResolveProviders+0x4e00 (74834000)
74827bdf je brcrypt!BCryptGenRandom+0x37 (74827bf7)
74827bd0 byte ptr [eax+10h],4
74827bd1 je brcrypt!BCryptGenRandom+0x37 (74827bf7)
74827bd2 push esi
74827bd3 push dword ptr [ebp+10h]
74827bd4 push dword ptr [ebp+0c]
74827bd5 push dword ptr [ebp+8]
74827bd6 push dword ptr [eax+14h]
74827bd7 push dword ptr [eax+10h]
74827bd8 push 20h
74827bd9 pop ecx
74827bda call brcrypt!BCryptGenRandom+0x2fda (74822fda)
74827bdc test esi,2
74827bde je brcrypt!BCryptGenRandom+0x6d (74827c2d)
74827bdf cmp dword ptr [ebp+8],0
74827be0 je brcrypt!BCryptGenRandom+0x51 (74827c11)
74827be1 mov esi,BC0000000h
74827be2 push 195Ah
74827be3 jmp brcrypt!BCryptGenRandom+0xbd (74827c7d)
74827be4 mov dword ptr [ebp+10h]
74827be5 and esi,0FFFFFFFh
74827be6 mov ecx,dword ptr [ebp+0ch]
74827be7 push esi
74827be8 call brcrypt!BCryptGetFipsAlgorithmNode+0x1e79 (7482b389)
74827be9 mov esi,eax
74827beb test esi,esi
74827bec jns brcrypt!BCryptGenRandom+0x0 (74827c90)
74827bed push 195Ah
74827bee jmp brcrypt!BCryptGenRandom+0xbd (74827c7d)
74827bef mov ecx,dword ptr [ebp+8]

```

Figure 2: Random Generation of Bytes

Since this API has been deprecated, I would expect the actors to change this in a future version of the ransomware, perhaps to the newer Cryptography API: Next Generation (CNG), which provides a new API `BCryptGenRandom` to achieve the same result.

A significant difference between the earlier Ryuk binary and our recent sample is the time it takes to fully encrypt the local disk. The 2018 binary takes close to one hour to encrypt the local disk, while the 2020 version takes less than 10 minutes.

Process Name	PID	Private Bytes	Working Set	Working Set Size	Path	Company Name
8d3f68b16f0710f858d0c...	10060	11.09	62.23 kB/s	3.9 MB	DESKTOP-FRH... \Marco	
net.exe	10028			1.23 MB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	2620			1.98 MB	DESKTOP-FRH... \Marco	Console Window Host
net1.exe	8704			1.22 MB	DESKTOP-FRH... \Marco	Net Command
net.exe	7060			1.2 MB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	10140			1.99 MB	DESKTOP-FRH... \Marco	Console Window Host
net.exe	8944			1.11 MB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	9720			2.04 MB	DESKTOP-FRH... \Marco	Console Window Host
net1.exe	2864			1.12 MB		
net.exe	6924			1.11 MB	DESKTOP-FRH... \Marco	
conhost.exe	5472			2 MB		
net1.exe	7708			948 kB		
net.exe	6440			512 kB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	7692			1.73 MB	DESKTOP-FRH... \Marco	Console Window Host
net.exe	4720			512 kB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	4932			1.57 MB	DESKTOP-FRH... \Marco	Console Window Host
net.exe	9588			516 kB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	7048			1.02 MB	DESKTOP-FRH... \Marco	Console Window Host
net.exe	4944			1.11 MB	DESKTOP-FRH... \Marco	Net Command
conhost.exe	9580			1.96		
net1.exe	4852			1.12		
net.exe	3604			1.11		
conhost.exe	4204			2		
net1.exe	3560			540 KB	DESKTOP-FRH... \Marco	

57:13:09

Figure 3: 2018 Ryuk 2018 Slower Encryption

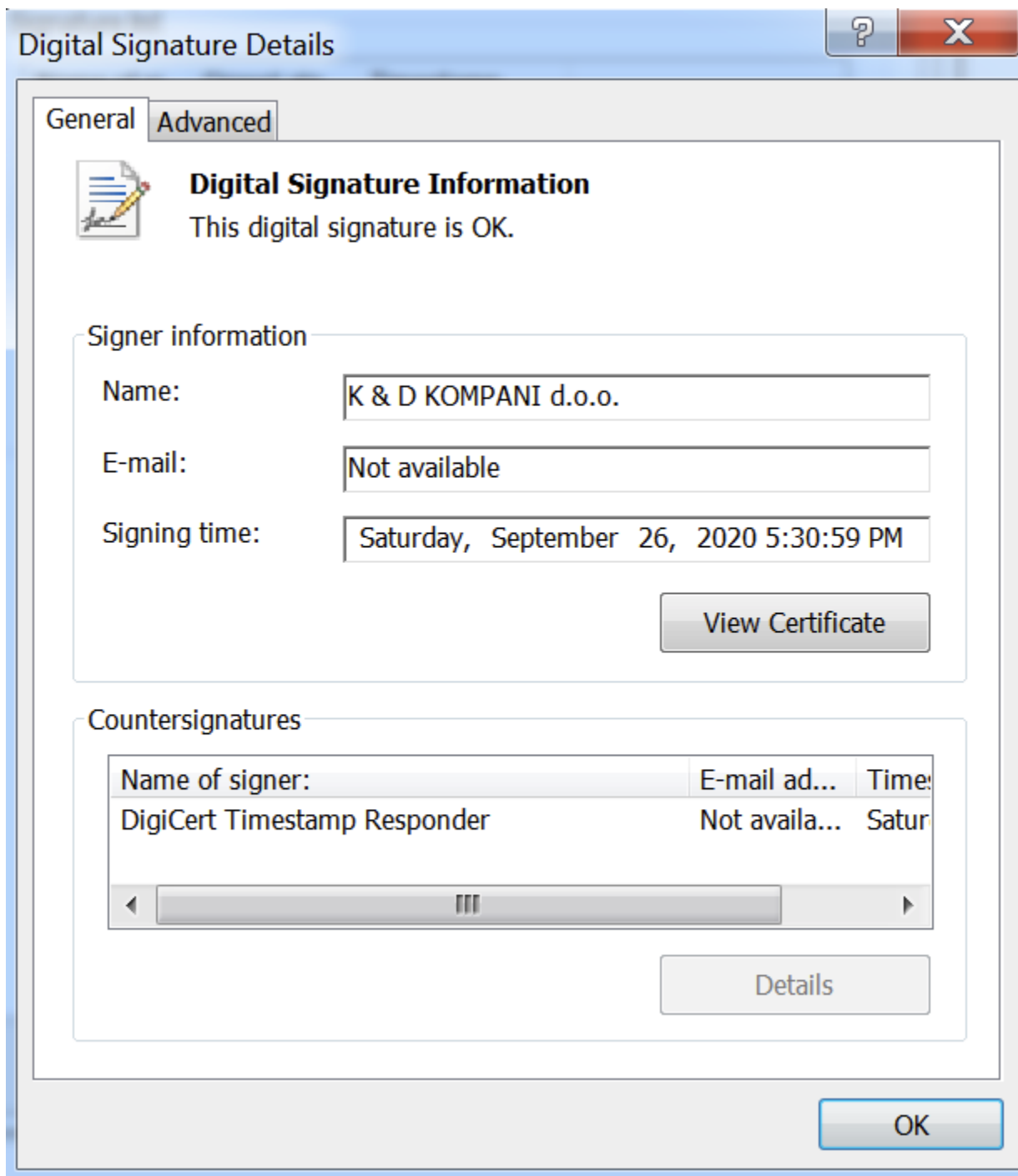
The increased encryption speed of the newer Ryuk variant places an extra burden on enterprise security efforts. The reaction time to detect, mitigate, and eradicate Ryuk before major damage is done is significantly limited, and many organizations have been unable to contain the ransomware in time. This occurred in UHS networks and required hospital staffers to shut down computer systems immediately to prevent further machines becoming infected by Ryuk.

## Diving Deeper into Ryuk 2020

---

This particular Ryuk sample (f8bc1638ec3b04412f708233e8586e1d91f18f6715d68cba1a491d4a7f457da0) has a signed digital certificate which was explicitly revoked by its issuer.

- Serial Number: 0a 1d c9 9e 4d 52 64 c4 5a 50 90 f9 32 42 a3 0a
- Subject: CN = K & D KOMPANI d.o.o



Figure

#### 4: 2020 Ryuk Revoked Certificate

When Ryuk begins executing, it duplicates itself and dumps this copy into the same directory with a randomly generated 8 character name. However, the file name always ends with "...lan.exe". These duplicate files help to start multiple threads. Ryuk utilizes a list of hardcoded strings to search for and stop specific running processes (Figure 1). It then tries to inject itself into additional processes.



f8bc1638ec3b04412f708233e8586e1d91f18f...	10/5/2020 6:33 PM	Application	377 KB
GUSicTuVqlan.exe	10/5/2020 6:33 PM	Application	377 KB
TDdkcGWMzlan.exe	10/5/2020 6:33 PM	Application	377 KB
WKgOBEziwlan.exe	10/5/2020 6:33 PM	Application	377 KB

```
L"C:\\Users\\NTK\\Desktop\\New folder\\GUSicTuVqlan.exe"
&L"C:\\Users\\NTK\\Desktop\\New folder\\GUSicTuVqlan.exe" 8 LAN"
&L"C:\\Users\\NTK\\Desktop\\New folder\\GUSicTuVqlan.exe"
&L"C:\\Users\\NTK\\Desktop\\New folder\\GUSicTuVqlan.exe" 8 LAN"
```

Figure 5: Droppers

Ryuk next begins executing certain command line tools to achieve some of its devastating effects; in particular, it tries to prevent user recovery by attempting to delete the Volume Shadow copies by leveraging `cmd.exe /c 'wmic.exe shadowcopy delete'`. This is followed with a `cmd.exe /c 'vssadmin.exe Shadows /all /quiet'` and `cmd.exe /c 'bcdedit /set {default} recoveryenabled No & bcdedit /set {default}'`.

```
ntdll.771C4267
mov dword ptr ss:[ebp-120],ntdll.770F77D8 ; 770F77D8:L"svchost.exe"
mov dword ptr ss:[ebp-11C],ntdll.770F77F0
xor esi,esi
mov dword ptr ss:[ebp-118],ntdll.770F7814 ; 770F7814:L"csrss.exe"
mov dword ptr ss:[ebp-114],ntdll.770F7828 ; 770F7828:L"smss.exe"
mov dword ptr ss:[ebp-110],ntdll.770F783C ; 770F783C:L"services.exe"
mov dword ptr ss:[ebp-10C],ntdll.770F7858 ; 770F7858:L"lsass.exe"
call <ntdll.RtlGetSuiteMask>
test eax,10000
jne ntdll.771C43C1
```

Figure 6:

Services that are not stopped

An `icacls.exe` is created in the Windows WoW directory, which gives the group **Everyone** full permissions to the drives on the system so that Ryuk has everything it needs to encrypt all drives.

```
."icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
"icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
'icacls "C:\\*" / grant Everyone:F /T /C /Q"
```

Figure 7a: grant permissions

```
msvcrt.74935B57
mov dword ptr ds:[<_acmdl>],eax ; 749B5BA0:&"icacls "C:\\*" /grant Everyone:F /T /C /Q"
call <JMP.&GetCommandLine>
mov dword ptr ds:[<_wcmdln>],eax
call msvcrt.7494E415
call msvcrt.74967D56
test eax,eax
js msvcrt.74935B8B
```

Figure 7b: icacls permissions granted

## Extracting the Executable from Memory

To avoid detection, the malware uses various evasion techniques like self injection. Ryuk uses this technique by allocating memory in which it writes a PE file. After this, it calls `VirtualProtect` to change the execution permissions on the section.

A fast way to extract the executable from memory is to run the binary in a debugger and set a breakpoint at the location of the allocated memory. For this, I use `x32dbg`, and set a breakpoint on `VirtualAlloc`. A thing to note is that when setting a breakpoint for `VirtualAlloc` you should follow the `jmp` routine into `Kernelbase` to get the base address of the newly allocated region and set the breakpoint on the return. Once the debugger has run, the breakpoint will hit. Follow the `EAX` register to the memory dump section to view if the `MZ` magic is present.

The screenshot displays the x32dbg debugger interface. At the top, the 'Breakpoints' window shows a breakpoint set on `kernelbase.dll` at address `7696F84C`, with the instruction `ret 10`. A red arrow labeled 'BP VirtualAlloc' points to this entry. The main disassembly window shows the code at this address, with `ret 10` highlighted. A red arrow labeled 'BP Hit Follow In Dump' points to the `ret 10` instruction. The right-hand pane shows the register dump, with `EAX` containing `00100000`. Below the disassembly, the memory dump window shows the hex and ASCII values of the memory, with a red arrow pointing to the 'MZ' magic bytes at address `00100000`.

Figure 8: Dumping the Binary



When the process is run, it will hit the breakpoint `VirtualAlloc` and in the **EAX** is the newly allocated virtual memory section to begin loading a copy of itself into this section. Following the **EAX** to the memory dump shows that the memory has been allocated for loading. When continuing the process, the dump window begins to populate with data as it hits the breakpoint that is set several times. Once it is confirmed that the binary has been fully loaded into this section, the binary data can be dumped for inspection.

Hex	ASCII
00 00 5F 5F 76 65 63 74 6F 72 63 61 6C 6C 00 00	...__vectorcall..
00 00 5F 5F 63 6C 72 63 61 6C 6C 00 00 00 5F 5F	...__clrcall...__
65 61 62 69 00 00 5F 5F 70 74 72 36 34 00 5F 5F	eabi...__ptr64...
72 65 73 74 72 69 63 74 00 00 5F 5F 75 6E 61 6C	restrict...__unal
69 67 6E 65 64 00 72 65 73 74 72 69 63 74 28 00	igned.restrict(. .. new.... delet
00 00 20 6E 65 77 00 00 00 00 20 64 65 6C 65 74	.. new.... delet
65 00 3D 00 00 00 3E 3E 00 00 3C 3C 00 00 21 00	e.=...>>...<<...!
00 00 3D 3D 00 00 21 3D 00 00 5B 5D 00 00 6F 70	...==...!=...[]..op
65 72 61 74 6F 72 00 00 00 00 2D 3E 00 00 2A 00	erator...->...*. .. ++...--...+.
00 00 2B 2B 00 00 2D 2D 00 00 2D 00 00 00 2B 00	.. ++...--...+.
00 00 26 00 00 00 2D 3E 2A 00 2F 00 00 00 25 00	..&...->*.../...%.
00 00 3C 00 00 00 3C 3D 00 00 3E 00 00 00 3E 3D	..<...<=...>...>=
00 00 2C 00 00 00 28 29 00 00 7E 00 00 00 5E 00	...()...~...^.
00 00 7C 00 00 00 26 26 00 00 7C 7C 00 00 2A 3D	.. ...&&... ...*=
00 00 2B 3D 00 00 2D 3D 00 00 2F 3D 00 00 25 3D	..+=...-=.../=...%=
00 00 3E 3E 3D 00 3C 3C 3D 00 26 3D 00 00 7C 3D	..>>=...<<=...&=... =
00 00 5E 3D 00 00 60 76 66 74 61 62 6C 65 27 00	..^=...vftable'. ..vhtable'...v
00 00 60 76 62 74 61 62 6C 65 27 00 00 00 60 76	..vhtable'...v
63 61 6C 6C 27 00 60 74 79 70 65 6F 66 27 00 00	call'. 'typeof'. .. local static
00 00 60 6C 6F 63 61 6C 20 73 74 61 74 69 63 20	.. local static
67 75 61 72 64 27 00 00 00 60 73 74 72 69 6E	guard'.... string
67 27 00 00 64 27 00 00 62 61 73 65 20 64 65 73	g'.... vbase des
74 72 75 63 74 6F 72 27 00 00 60 76 65 63 74 6F	tructor'.. vecto
72 20 64 65 6C 65 74 69 6E 67 20 64 65 73 74 72	r deleting destr
75 63 74 6F 72 27 00 00 00 60 64 65 66 61 75	uctor'.... defau
6C 74 20 63 6F 6E 73 74 72 75 63 74 6F 72 20 63	lt constructor c
6C 6F 73 75 72 65 27 00 00 60 73 63 61 6C 61	losure'.... scala
72 20 64 65 6C 65 74 69 6E 67 20 64 65 73 74 72	r deleting destr
75 63 74 6F 72 27 00 00 00 60 76 65 63 74 6F	uctor'.... vecto
72 20 63 6F 6E 73 74 72 75 63 74 6F 72 20 69 74	r constructor it
65 72 61 74 6F 72 27 00 00 60 76 65 63 74 6F	erator'.... vecto
72 20 64 65 73 74 72 75 63 63 74 6F 72 20 69 74	r destructor ite
72 61 74 6F 72 27 00 00 60 76 65 63 74 6F	erator'.... vecto
72 20 76 62 61 73 65 20 63 6F 6E 73 74 72 75 63	r vbase construc
74 6F 72 20 69 74 65 72 61 74 6F 72 27 00 60 76	tor iterator'. v
69 72 74 75 61 6C 20 64 69 73 70 6C 61 63 65 6D	irtual displacem
65 6E 74 20 6D 61 70 27 00 00 60 65 68 20 76 65	ent map'.. eh ve
63 74 6F 72 20 63 6F 6E 73 74 72 75 63 74 6F 72	ctor constructor
20 69 74 65 72 61 74 6F 72 27 00 00 60 65	iterator'.... e
68 20 76 65 63 74 6F 72 20 64 65 73 74 72 75 63	h vector destr
74 6F 72 20 69 74 65 72 61 74 6F 72 27 00 60 65	tor iterator'. e
68 20 76 65 63 74 6F 72 20 76 62 61 73 65 20 63	h vector vbase c
6F 6E 73 74 72 75 63 74 6F 72 20 69 74 65 72 61	onstructor ite
74 6F 72 27 00 00 60 63 6F 70 79 20 63 6F 6E 73	tor'.... copy cons
74 72 75 63 74 6F 72 20 63 6C 6F 73 75 72 65 27	tructor closure'
00 00 60 75 64 74 20 72 65 74 75 72 6E 69 6E 67	..udt returning
27 00 60 45 48 00 60 52 54 54 49 00 00 60 6C	'..EH..RTTI...l
6F 63 61 6C 20 76 66 74 61 62 6C 65 27 00 60 6C	ocal vftable'.l
6F 63 61 6C 20 76 66 74 61 62 6C 65 20 63 6F 6E	ocal vftable con
73 74 72 75 63 74 6F 72 20 63 6C 6F 73 75 72 65	structor closure
27 00 20 6E 65 77 58 5D 00 00 20 64 65 6C 65 74	'.. new[].. delet
65 58 5D 00 00 60 6F 6D 6E 69 20 63 61 6C 6C	e[...].. omni call
73 69 67 27 00 00 60 70 6C 61 63 65 6D 65 6E 74	sig'.. placem
20 64 65 6C 65 74 65 20 63 6C 6F 73 75 72 65 27	'.. delete closure'
00 00 60 70 6C 61 63 65 6D 65 6E 74 20 64 65 6C	.. 'placement del
65 74 65 58 5D 20 63 6F 6F 73 75 72 65 27 00 00	ete[] closure'..
00 00 60 6D 61 6E 61 67 65 64 20 76 65 63 74 6F	.. 'managed vecto
72 20 63 6F 6E 73 74 72 75 63 74 6F 72 20 69 74	r constructor it
65 72 61 74 6F 72 27 00 00 60 6D 61 6E 61 67	erator'.... manag
65 64 20 76 65 63 74 6F 72 20 64 65 73 74 72 75	ed vector destr
63 74 6F 72 20 69 74 65 72 61 74 6F 72 27 00 00	ctor iterator'..
00 00 60 65 68 20 76 65 63 74 6F 72 20 63 6F 70	.. 'eh vector cop
79 20 63 6F 6E 73 74 72 75 63 74 6F 72 20 69 74	y constructor it
65 72 61 74 6F 72 27 00 00 60 65 68 20 76 65	erator'.... 'eh ve
63 74 6F 72 20 76 62 61 73 65 20 63 6F 70 79 20	ctor vbase copy
63 6F 6E 73 74 72 75 63 74 6F 72 20 69 74 65 72	constructor iter
61 74 6F 72 27 00 60 64 79 6E 61 6D 69 63 20 69	ator'. 'dynamic i
6E 69 74 69 61 6C 69 74 65 72 20 66 6F 72 20 27	nitializer for '
00 00 60 64 79 6E 61 6D 69 63 20 61 74 65 78 69	.. 'dynamic atexi
74 20 64 65 73 74 72 75 63 74 6F 72 20 66 6F 72	t destructor for
20 27 00 00 00 00 60 76 65 63 74 6F 72 20 63 6F	'.... vector co
70 79 20 63 6F 6E 73 74 72 75 63 74 6F 72 20 69	py constructor i
74 65 72 61 74 6F 72 27 00 00 60 76 65 63 74 6F	erator'.. vecto
72 20 76 62 61 73 65 20 63 6F 70 79 20 63 6F 6E	r vbase copy con
73 74 72 75 63 74 6F 72 20 69 74 65 72 61 74 6F	structor ite
72 27 00 00 00 00 60 6D 61 6E 61 67 65 64 20 76	r'.... 'managed v
65 63 74 6F 72 20 63 6F 70 79 20 63 6F 6E 73 74	ector copy const
72 75 63 74 6F 72 20 69 74 65 72 61 74 6F 72 27	ruktor iterator'
00 00 60 6C 6F 63 61 6C 20 73 74 61 74 69 63 20	.. 'local static
74 68 72 65 61 64 20 67 75 61 72 64 27 00 6F 70	hread guard'.op
65 72 61 74 6F 72 20 22 20 00 00 00 00 20 54	erator "" .... T
79 70 65 20 44 65 73 63 72 69 70 74 6F 72 27 00	ype Descriptor'. .. Base Class De
00 00 20 42 61 73 65 20 43 6C 61 73 73 20 44 65	.. Base Class De
73 63 72 69 70 74 6F 72 20 61 74 20 28 00 20 42	scriptor at (. B
61 73 65 20 43 6C 61 73 73 20 41 72 72 61 79 27	ase Class Array'
00 00 20 43 6C 61 73 73 20 48 69 65 72 61 72 63	.. Class Hierarc
68 79 20 44 65 73 63 72 69 70 74 6F 72 27 00 00	hy Descriptor'..
00 00 20 43 6F 6D 70 6C 65 74 65 20 4F 62 6A 65	.. Complete Obje

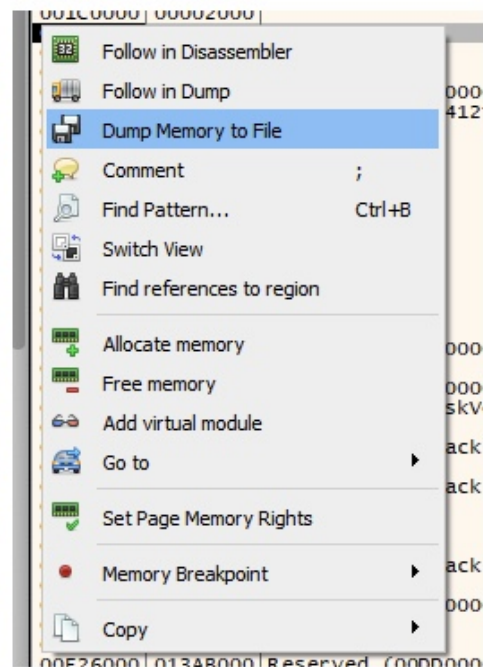


Figure 9: Dump Memory To File



The next step is to right-click on the memory dump and follow the dump in the memory map. This brings you to where the dump has been allocated in memory, and from here you can dump the memory to file. However, as shown in Figure 9, notice that the dumped memory does not have a valid PE header; we have to modify the header so the PE (Figure 10) can work in the tool of your choice.

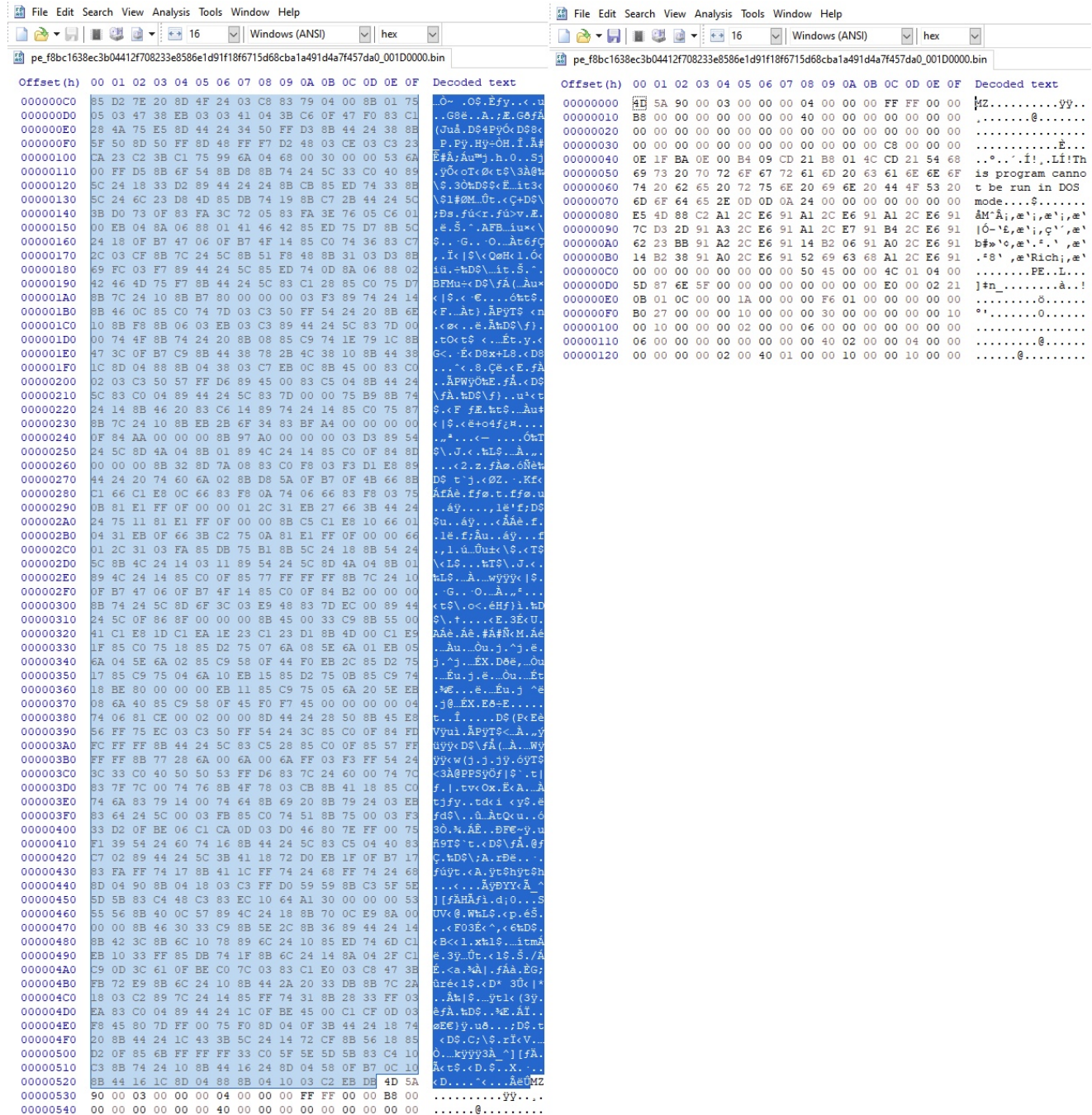


Figure 10: Fix the Headers

This particular binary is straightforward to modify. Open up your favorite Hex editor, load the file, highlight everything before the **MZ** and delete it. Sometimes, just deleting extra bytes will not work if a blob of memory has corrupted magic bytes. In that scenario, you can copy a known good header and add it to the corrupted PE header to make a valid PE.

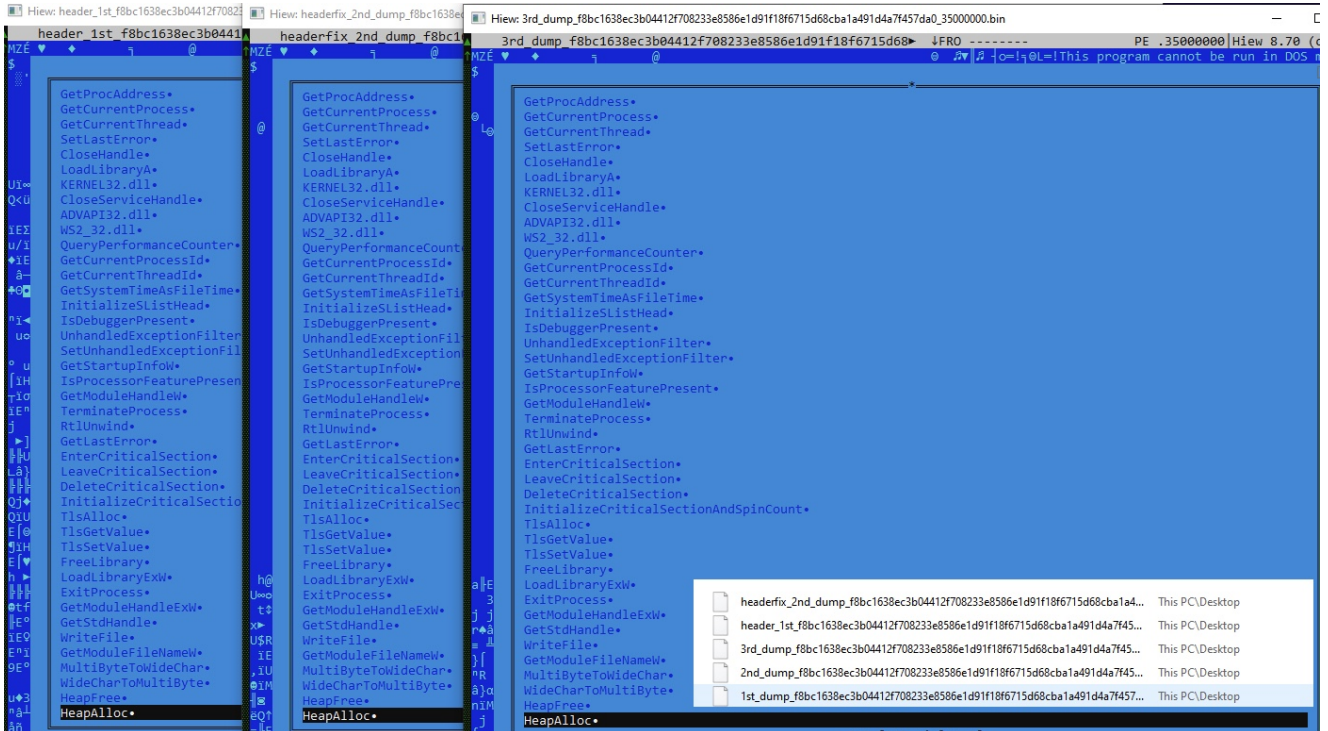


Figure 11: Three Memory Dump Files

If you follow the process from the beginning, the breakpoint will hit `VirtualAlloc` additional times. I've dumped the memory with the techniques shown above to show why Ryuk's encryption on a system is so fast:

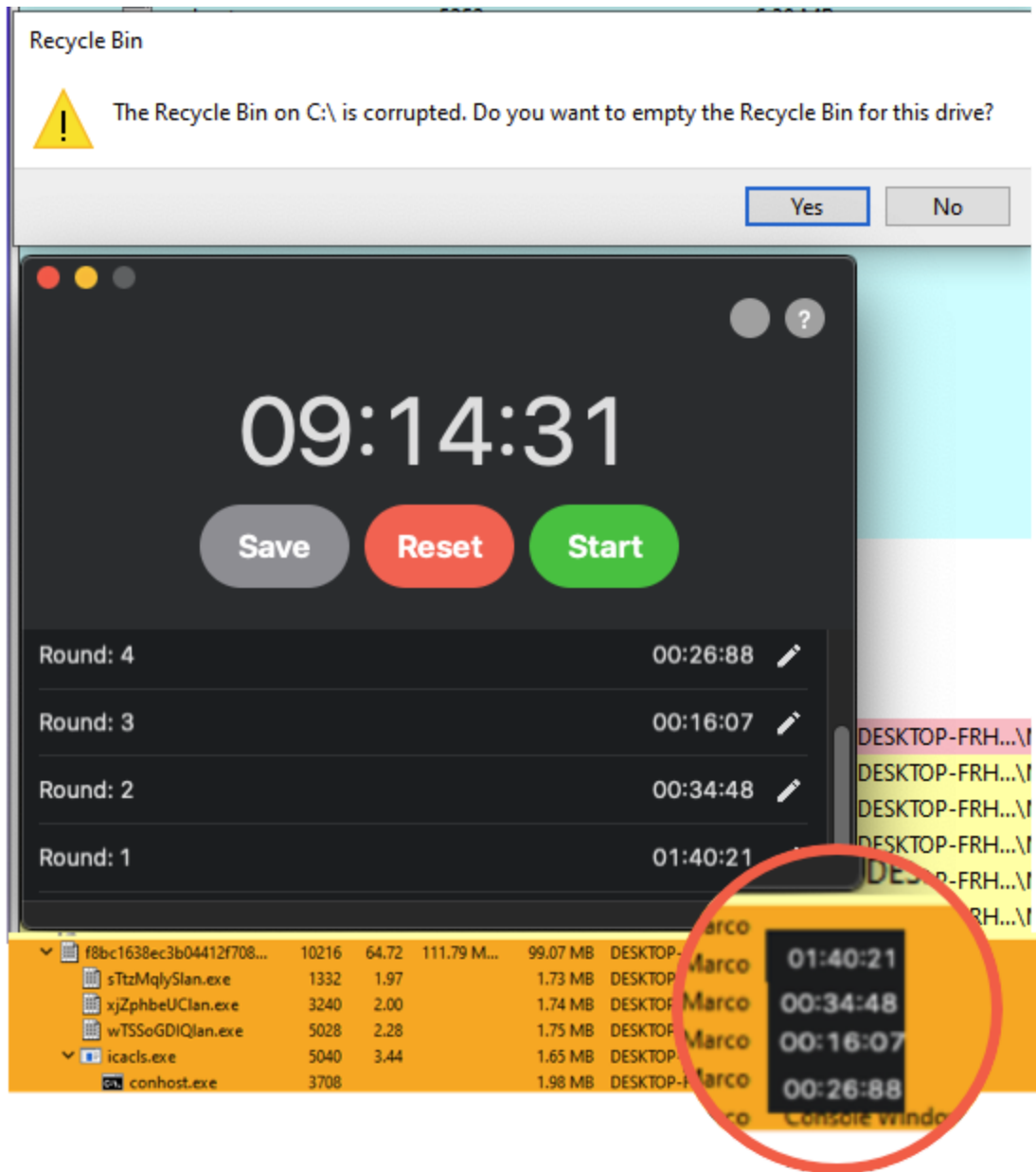


Figure 12:

Speed of Ryuk 2020

## Conclusion

The FBI has stated that Ryuk Ransomware actors have been paid over 61 million dollars. With Ryuk attacks crippling organizations, this number will soon surpass the 100 million mark if it hasn't done so already.

However, guarding against the ransomware menace in general and Ryuk in particular is not complicated with the proper protection in place: The techniques used by these cybercriminals are well-understood and relatively simple. The weaknesses they exploit are organizations' inability to detect and remediate at speed, but this is a problem that can be and has been solved.



Meanwhile, as analysts, it's important that we keep up with the latest developments and techniques deployed by adversaries. At SentinelOne, we track the ever-changing variants of Ryuk to understand the latest capabilities added to this ransomware family. In this post, we have detailed how Ryuk has evolved to increase its speed of encryption and the methods it uses for evasion. In a future post, we will cover Ryuk's network layer and the many artifacts collected during our analysis process.

## Samples

---

SHA256: f8bc1638ec3b04412f708233e8586e1d91f18f6715d68cba1a491d4a7f457da0

SHA1: c3fa91438850c88c81c0712204a273e382d8fa7b

SHA256: 7e28426e89e79e20a6d9b1913ca323f112868e597fc6b9e073102e73407b47

SHA1: 5767653494d05b3f3f38f1662a63335d09ae6489

## MITRE ATT&CK

---

Command and Scripting Interpreter [T1059](#)

Native API [T1106](#)

Application Shimming [T1546.011](#)

Process Injection [T1055](#)

Masquerading [T1036](#)

Virtualization/Sandbox Evasion [T1497.001](#)

Deobfuscate/ Decode Files [T1140](#)

Obfuscated Files or Information [T1027](#)

System Time Discovery [T1124](#)

Security Software Discovery [T1518.001](#)

Process Discovery [T1057](#)

File and Directory Discovery [T1083](#)

System Information Discovery [T1082](#)

Archive Collected Data [T1560](#)

Encrypted Channel [T1573](#)