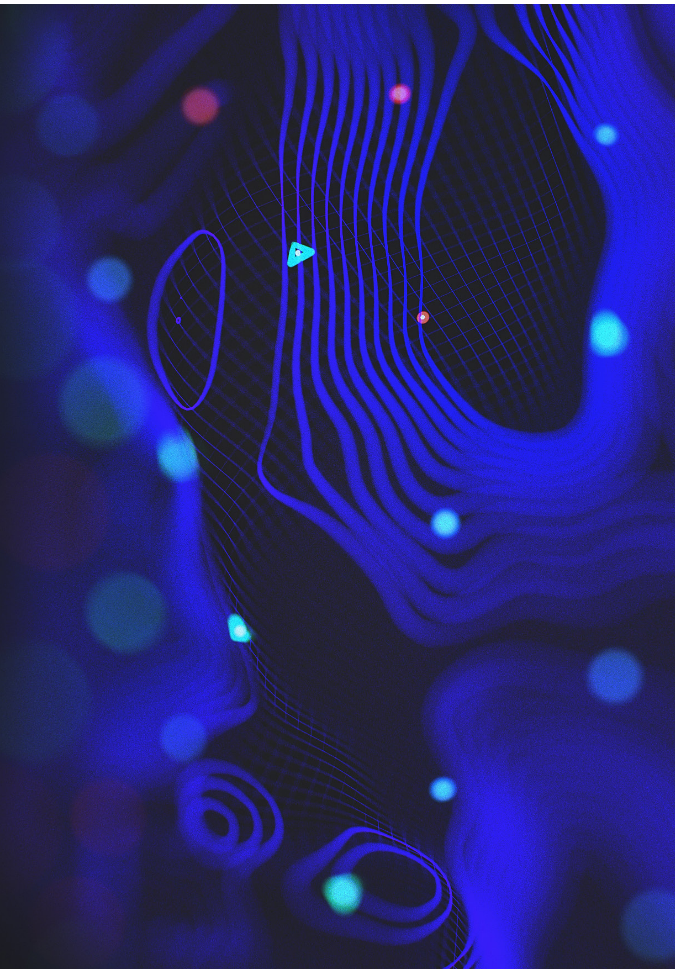
# CONTI UNPACKED: UNDERSTANDING RANSOMWARE DEVELOPMENT AS A RESPONSE TO DETECTION – A DETAILED TECHNICAL ANALYSIS

Authors: Idan Weizman, Antonio Pirozzi

July 2021

SentinelLABS Research Team

# TABLE OF CONTENTS



# INTRODUCTION

Not yet two years old and already in its seventh iteration, Ransomware as a Service variant Conti has proven to be an agile and adept malware threat, capable of both autonomous and guided operation and with unparalleled encryption speed. As of June 2021, Conti's unique feature set has helped its affiliates to extort several million dollars from over 400 organizations. In this report, we describe in unprecedented detail the rapid evolution of this ransomware and how it has adapted quickly to defenders' attempts to detect and analyze it.

SentinelLabs Team

## CONTI BACKGROUND

Conti is developed and maintained by the so-called TrickBot gang, and it is mainly operated through a RaaS affiliation model. The Conti ransomware is derived from the codebase of Ryuk and relies on the same TrickBot infrastructure.

Conti samples first began to be seen around October 2019. Recent attacks, such as that on Ireland's public health service, demonstrate that Conti has succeeded in becoming just as dangerous if not more so than its predecessor, for both organizations and the public at large. There are 399 reported Conti incidents at the time of writing:

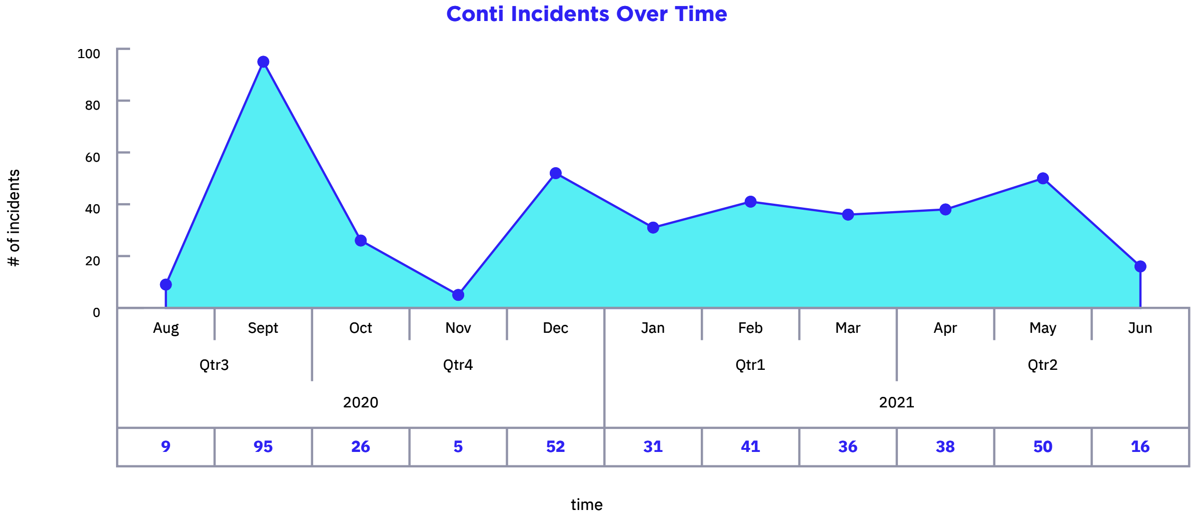
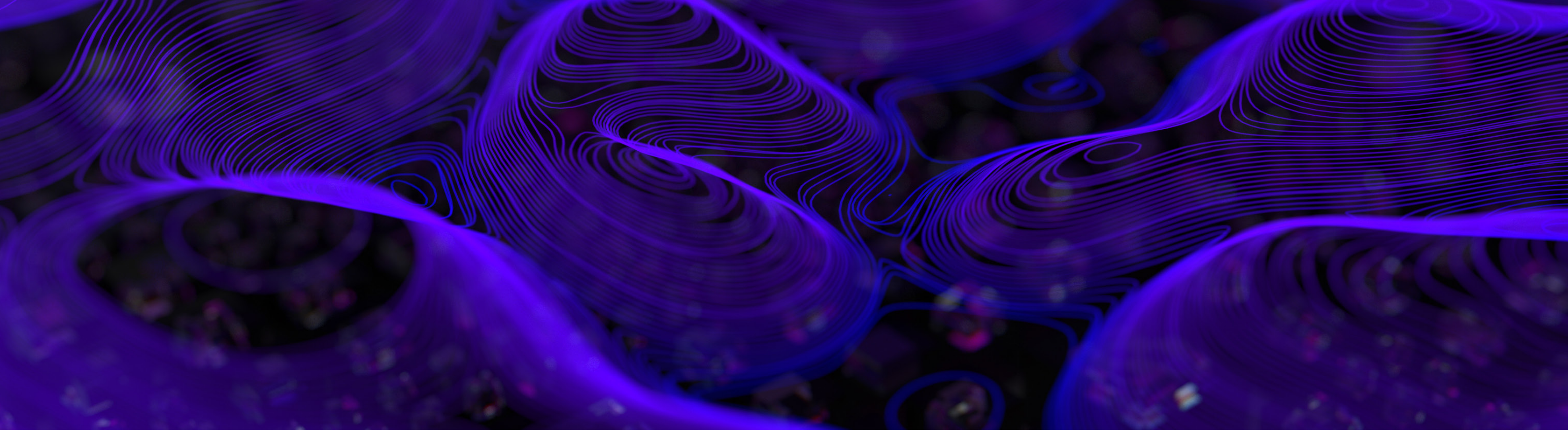


Fig 1: Source DarkTracer

In common with many other ransomware families, Conti also operates a leaks site in order to put further pressure on its victims to pay.

Historically, ransomware authors have built their tools adopting an iterative process, evolving and continuously adapting them to make them more effective and evasive. This has happened since Dyre (2014), and it's still true in Conti, which looks different in each iteration. This development model proves effective in managing the malware's stealthiness. With each iteration, the ransomware signature changes because all the static indicators are encrypted using a different logic.



This technical analysis aims to outline the Conti phylogenesis since the ransomware first appeared on the scene, in order to build a comprehensive knowledge of Conti's evolution and its development pipeline.

For this study, we clustered Conti samples by timestamps. All the samples used in this research are readily available from OSINT and are recognized as Conti both by the community and by static and dynamic analysis done herein. We found that each iteration implemented new features in Conti and evolved existing ones. The following key ransomware characteristics are summarized:

- **Obfuscation improvements:** Since the early 'test samples' (late 2019), Conti started implementing a simple XOR mechanism to hide the API names resolved at runtime. From June 2020, a custom encoding function for string obfuscation was also employed.

- **Focus on speed:** Conti developers focus mainly on speed; Conti uses up to 32 concurrent CPU threads for file encryption operations and, starting from the iteration of September 2020, they switched the encryption algorithm from AES to CHACHA in order to further speed-up the encryption process. Efficiency seems to be a functional requirement in the Conti development process. This translates into less time required to lock victim's data and reduce the chance of the operation being blocked.

- **Optimization of file encryption:** from 2nd September 2020, a new logic for file encryption was added. The logic implements two different modes: full and partial. depending on file extension and file dimension. Moreover, encryption through IoCompletionPorts was replaced by C++ queues and locks in January 2021.

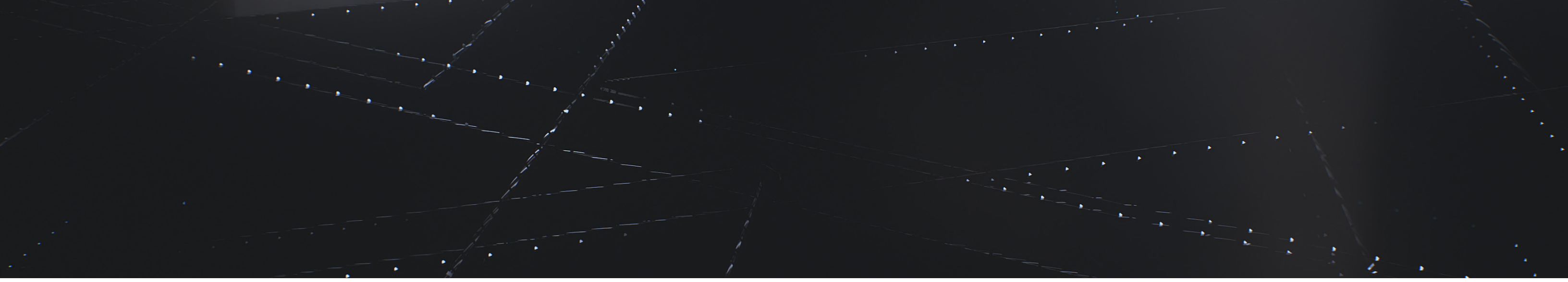
## OPERATIONAL BACKGROUND

Conti shares some code with Ryuk version 2 and uses the same template for the ransom note. Starting from June 2018, Ryuk began to be deployed by the Trickbot loader, which was spread through Emotet. Such attacks begin with Emotet used as a downloader for Trickbot, which then installs tools such as Cobalt Strike in order to gain full access to victim infrastructure. The delivery of Cobalt Strike is human-operated, and the objective is to drop Ryuk at the end of a very long attack chain, often extending over several weeks.

Initially, Ryuk and later Conti were delivered exclusively by TrickBot. However, by March 2021, as detections for TrickBot improved, BazarLoader/BazarBackdoor began to be used as the tool of choice for the delivery of Conti.

Conti today is sold behind a RaaS affiliation model and operated by different threat actors. It is developed and maintained by the so-called Trickbot gang.

Conti acts like most ransomware, but it has been engineered to be more efficient and evasive. One of the major ransomware trends today is the adoption of multithreading to speed up file encryption (like Revil, Cerber and Babuk), but Conti takes it to an even higher level. Conti uses 32 concurrent CPU threads to hasten the encryption operations, aiming to do maximum harm before being identified by endpoint security products. Moreover, the usage of a runtime API loading mechanism, an added layer of obfuscation by API search by hashing, and API-unhooking mechanism built inside to disable EDR-based API hooks, make this ransomware one of the most efficient and evasive in the current threat landscape.



## DEEP DIVE INTO THE DEVELOPMENT PROCESS

### In the Beginning

Most software development starts with a prototype, test version, or a proof of concept, and so does this ransomware. The earliest sample of Conti we could find[1] dates from the end of 2019 and includes indications that it is an early test version (e.g., the ransom note contains the text "test note"). It took eight months for this version to make headlines[2]. We should start our story here, with this early test version, in order to better learn about the development the ransomware went through.

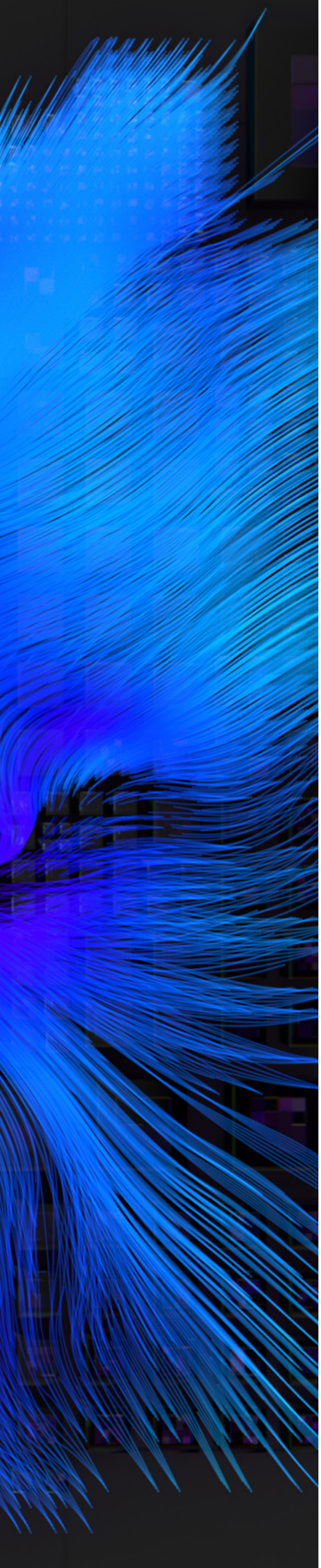
The Conti samples are sorted by the timestamp present in each sample PE file. Although this metadata can be altered easily, we didn't find any reason to believe it was modified, as the timestamps in newly discovered samples advance at the same rate as the date at which the samples are uploaded to online resources.

| Name | Hash (SHA-256) | Is Packed? | Timestamp (UTC) | Type |
|---|---|---|---|---|
| A1a | 2f334c0802147aa0eee90ff0a2b0e102<br>2325b5cba5cb5236ed3717a2b0582a9c | Yes | 2019/10/06 14:08:28 | EXE |
| A1b | 4f43a66d96270773f4e849055a844feb<br>6ef234d7340b797f8763b7a9f8d80583 | Yes | 2019/10/06 12:43:23 | DLL |
| A1c | 94bdec109405050d31c2748fe3db32a<br>357f554a441e0eae0af015e8b6461553e | No | 2019/10/21 15:00:01 | EXE |
| A2 | 77b1fcae9e8f0a5a739c35961382e2b3f2<br>39a05c1135c4a8efe1964a263d5a47 | No | 2019/10/21 15:00:01 | EXE |

The names in the table above are used for readability. The uppercase letter is shared between different samples which are considered to have the exact same functionality, although they might present insignificant differences (e.g. ransom note text, encryption key). Then, samples which share a number originate from the same sample, where the lowercase letter at the end shows the order of unpacking ('a' being the original sample, 'b' extracted from it, 'c' extracted from 'b', etc.).

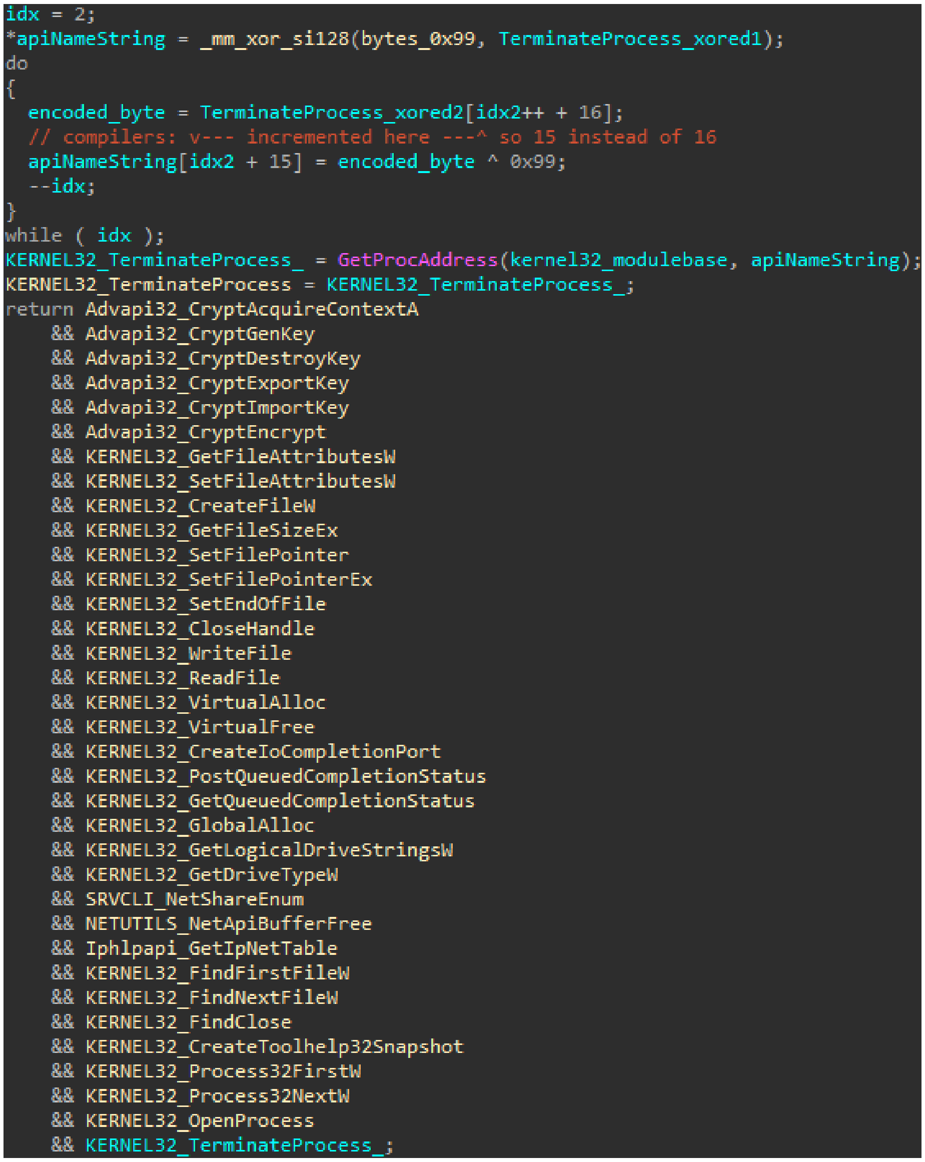
[1] https://id-ransomware.blogspot.com/2019/11/conti-ransomware.html

[2] https://www.carbonblack.com/blog/tau-threat-discovery-conti-ransomware/



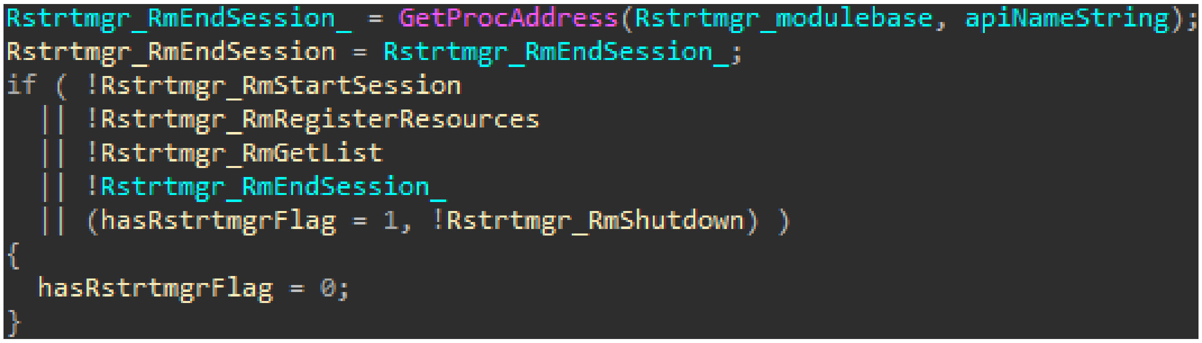
Conti has only a few of its imported functions linked at load-time. Therefore, the first thing it does is load them manually at runtime using LoadLibraryA and GetProcAddress.

All the names of the APIs are encoded using a simple XOR with the byte 0x99. The names of the DLLs are not encoded in this version, except for some optional imports, which are from Rstrtmgr.dll. Other than those, the GetProcAddress function ends by making sure it's got all the mandatory APIs it was looking for. Otherwise, it exits the program with ExitProcess.



```
idx = 2;
*apiNameString = _mm_xor_si128(bytes_0x99, TerminateProcess_xored1);
do
{
  encoded_byte = TerminateProcess_xored2[idx2++ + 16];
  // compilers: v--- incremented here ---^ so 15 instead of 16
  apiNameString[idx2 + 15] = encoded_byte ^ 0x99;
  --idx;
}
while ( idx );
KERNEL32_TerminateProcess_ = GetProcAddress(kernel32_modulebase, apiNameString);
KERNEL32_TerminateProcess = KERNEL32_TerminateProcess_;
return Advapi32_CryptAcquireContextA
    && Advapi32_CryptGenKey
    && Advapi32_CryptDestroyKey
    && Advapi32_CryptExportKey
    && Advapi32_CryptImportKey
    && Advapi32_CryptEncrypt
    && KERNEL32_GetFileAttributesW
    && KERNEL32_SetFileAttributesW
    && KERNEL32_CreateFileW
    && KERNEL32_GetFileSizeEx
    && KERNEL32_SetFilePointer
    && KERNEL32_SetFilePointerEx
    && KERNEL32_SetEndOfFile
    && KERNEL32_CloseHandle
    && KERNEL32_WriteFile
    && KERNEL32_ReadFile
    && KERNEL32_VirtualAlloc
    && KERNEL32_VirtualFree
    && KERNEL32_CreateIoCompletionPort
    && KERNEL32_PostQueuedCompletionStatus
    && KERNEL32_GetQueuedCompletionStatus
    && KERNEL32_GlobalAlloc
    && KERNEL32_GetLogicalDriveStringsW
    && KERNEL32_GetDriveTypeW
    && SRVCLI_NetShareEnum
    && NETUTILS_NetApiBufferFree
    && Iphlpapi_GetIpNetTable
    && KERNEL32_FindFirstFileW
    && KERNEL32_FindNextFileW
    && KERNEL32_FindClose
    && KERNEL32_CreateToolhelp32Snapshot
    && KERNEL32_Process32FirstW
    && KERNEL32_Process32NextW
    && KERNEL32_OpenProcess
    && KERNEL32_TerminateProcess_;
```

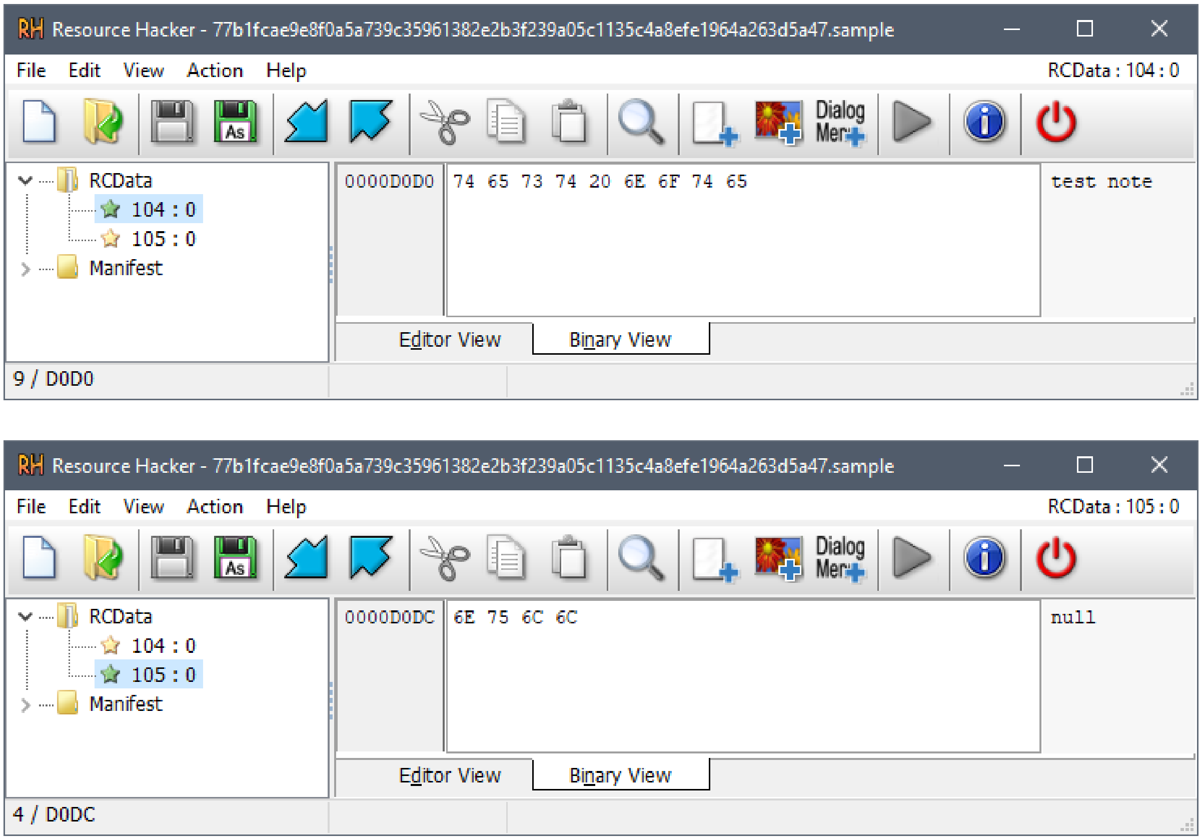
Fig 2: The imports loading function, getting the last import and checking all imports were found successfully



```
Rstrtmgr_RmEndSession_ = GetProcAddress(Rstrtmgr_modulebase, apiNameString);
Rstrtmgr_RmEndSession = Rstrtmgr_RmEndSession_;
if ( !Rstrtmgr_RmStartSession
  || !Rstrtmgr_RmRegisterResources
  || !Rstrtmgr_RmGetList
  || !Rstrtmgr_RmEndSession_
  || (hasRstrtmgrFlag = 1, !Rstrtmgr_RmShutdown) )
{
  hasRstrtmgrFlag = 0;
}
```

Fig 3: The import loading function, setting the flag if the optimal imports were found successfully

Two resources from the PE file are loaded. The first will be used as the text for the ransom note (which is set to “test note” in this earliest version), while the second is a list of strings, separated by a comma, of files that should be encrypted in case they contain a substring from the list. In cases where the resource has a value of “null”, all files are encrypted except for a hardcoded list. This allows for simple modifications to the readme text or for targeted encryption of specific files, without recompiling the ransomware.



Resource Hacker - 77b1fcae9e8f0a5a739c35961382e2b3f239a05c1135c4a8efe1964a263d5a47.sample

RCData : 104 : 0

```
0000D0D0  74 65 73 74 20 6E 6F 74 65          test note
```

9 / D0D0

Resource Hacker - 77b1fcae9e8f0a5a739c35961382e2b3f239a05c1135c4a8efe1964a263d5a47.sample

RCData : 105 : 0

```
0000D0DC  6E 75 6C 6C                         null
```

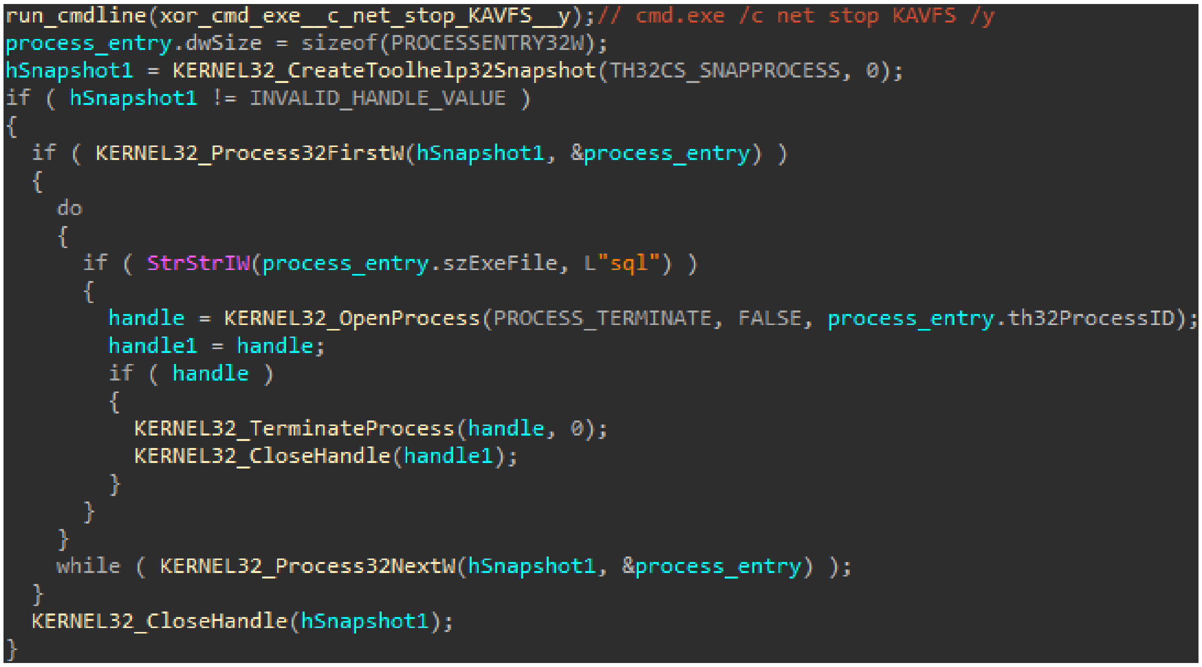
4 / D0DC

Fig 4 + 5: The ransom note and the null resources in the sample

A few operations are performed to increase the probability of successful encryption of the system:

- A list of command lines are executed: the first few are used to delete shadow copies over the first few drives in the system; the rest are to stop a list of ~170 services[3]. All command strings are encoded by XOR with the byte 0x99.

- All running processes on the system are iterated. Processes containing “sql” in them are terminated with TerminateProcess.



```
run_cmdline(xor_cmd_exe__c_net_stop_KAVFS__y);// cmd.exe /c net stop KAVFS /y
process_entry.dwSize = sizeof(PROCESSENTRY32W);
hSnapshot1 = KERNEL32_CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
if ( hSnapshot1 != INVALID_HANDLE_VALUE )
{
  if ( KERNEL32_Process32FirstW(hSnapshot1, &process_entry) )
  {
    do
    {
      if ( StrStrIW(process_entry.szExeFile, L"sql") )
      {
        handle = KERNEL32_OpenProcess(PROCESS_TERMINATE, FALSE, process_entry.th32ProcessID);
        handle1 = handle;
        if ( handle )
        {
          KERNEL32_TerminateProcess(handle, 0);
          KERNEL32_CloseHandle(handle1);
        }
      }
    }
    while ( KERNEL32_Process32NextW(hSnapshot1, &process_entry) );
  }
  KERNEL32_CloseHandle(hSnapshot1);
}
```

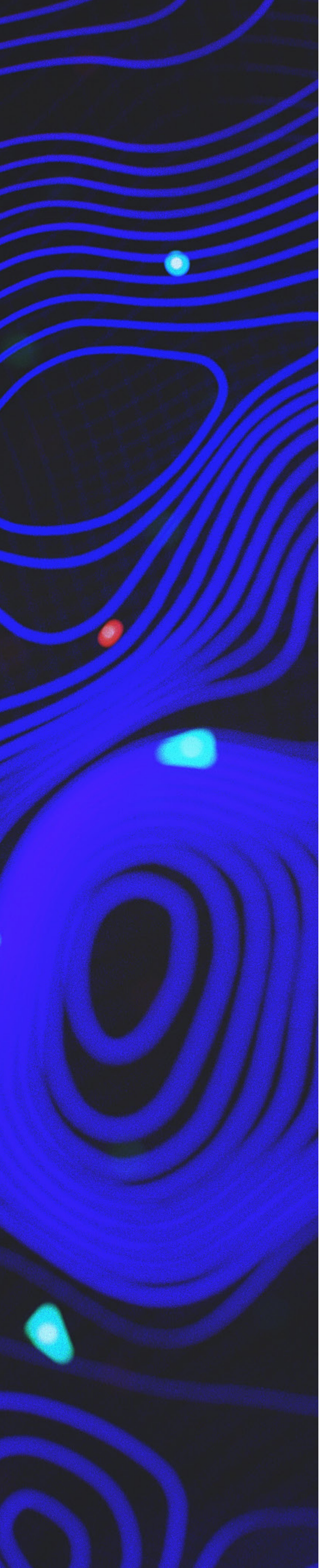
Fig 6: The last service stopped and the termination of processes containing “sql” in their name.

The amount of processors in the system is queried and twice that number of IoCompletionPorts are created (weirdly, this small detail will change quite a few times). The same amount of threads are created for handling the encryption of files, waiting for them to come through these IoCompletionPorts (detailed last as how those work).

Following the creation of the threads, the ransomware will look for all drives on the system, running its logic for selecting files for encryption on each of those.

After all the drives are iterated through, the ransomware will go over all addresses that are returned from GetIpNetTable, with 32 threads, and for each of those will call NetShareEnum to get a list of network shares opened on them. If any shares are found, they will be added to a list to traverse for files to encrypt as well.

[3] See Appendix A for the list of commands used.



Conti iterates on all found shares for files.

A marked CompletionStatus is sent, using the magic number ‘666’, for every created thread which iterates through the IoCompletionPorts, to inform them there are no more files left for encryption, causing them to stop.

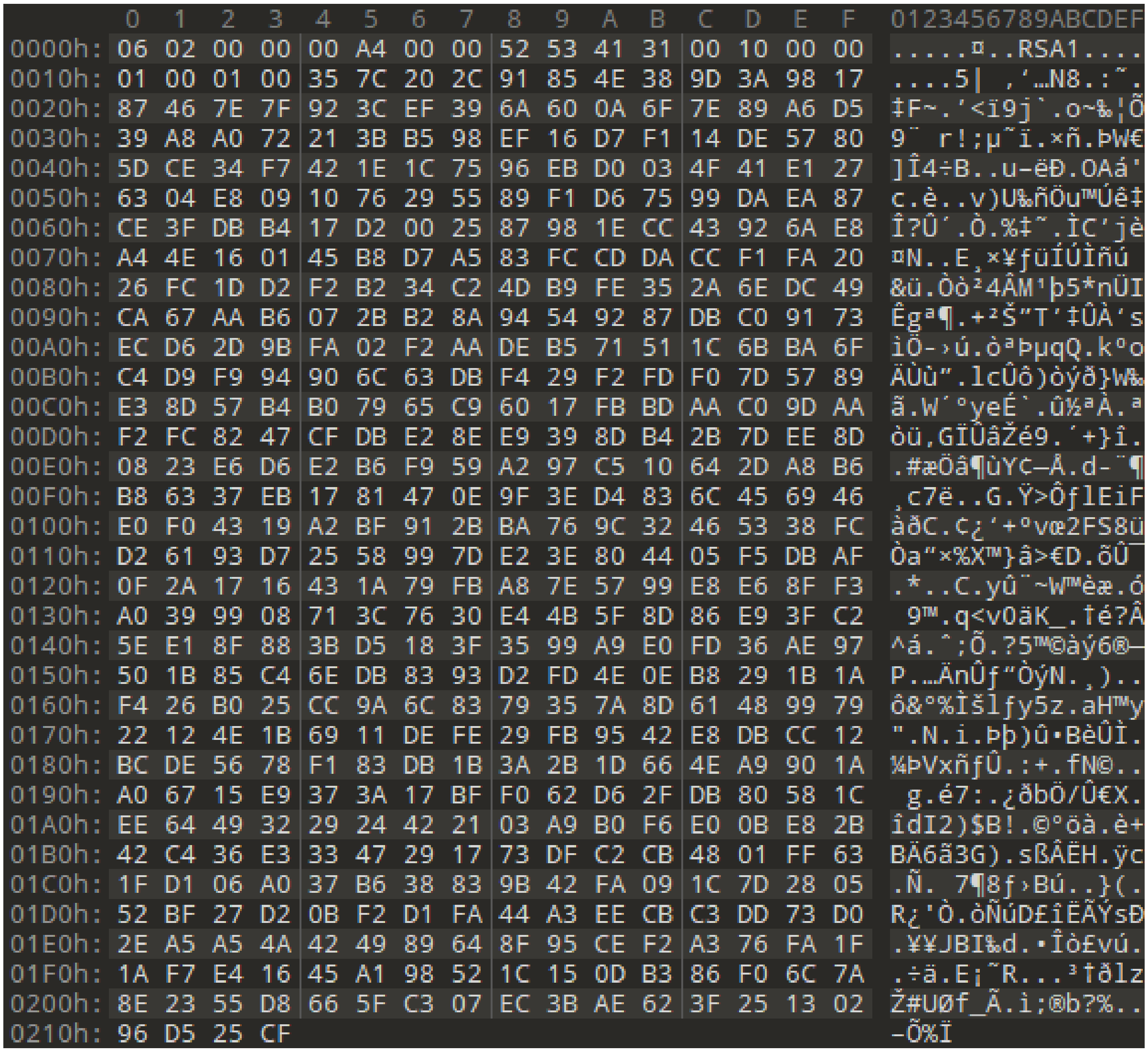
The two parts that are left to go over are the iteration function and the encryption worker thread function:

- The iteration function starts by getting a folder as a parameter, creating a readme file inside of it, as per the embedded resource. Afterward, using the FindFirstFile API, all entries in the folder are searched. Skipping the “.” and the “..” names, entries marked as directories will be recursively called as a parameter to the current function, except for an hardcoded list of discovery names which won’t be. As for files, except for a hardcoded list of substrings (executables, links, and conti created files), all other files will be queued for encryption.

- The encryption function each thread runs starts with loading a public RSA key placed in the data section of the PE file. After that, the function takes out items from the IoCompletionPorts, which contain a file path to encrypt, and does the following to each one: creates a new AES key for it, exported it using the public RSA key, tried to open the files for writing (we will note a few things about this part soon), writes the exported and encrypted key to the end of the given file, followed by the original size of the file (for allowing to separate the key from the data in it). Then, the content of the files is read, in chunks no bigger than 0x500000 bytes, encrypted with the AES key which was generated, and overwritten back to the file. At the end of the encryption process, the file will be moved, adding the extension “.CONTI” to the end of it, and the processes of getting files from the IoCompletionPorts will be repeated until the magic status (666) is sent.

The last thing to note is how file handles are opened by the encrypting function. Remember the optional imports from Rstrtmgr.dll? This is the place they are used. Those APIs are used by the Windows Restart Manager for asking opened processes to close themselves on restarts, which results in closing of open handles to files in the process. If Conti fails opening a file, and the optional imports are found, it will use them to force any process using the file to close. Afterward, Conti will retry opening the files.

When all threads return, the sample exits.



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 06 02 00 00 00 A4 00 00 52 53 41 31 00 10 00 00  .....¤..RSA1....
0010h: 01 00 01 00 35 7C 20 2C 91 85 4E 38 9D 3A 98 17  ....5| ,‘…N8.:˜.
0020h: 87 46 7E 7F 92 3C EF 39 6A 60 0A 6F 7E 89 A6 D5  ‡F~.’<ï9j`.o~‰¦Õ
0030h: 39 A8 A0 72 21 3B B5 98 EF 16 D7 F1 14 DE 57 80  9¨ r!;µ˜ï.×ñ.ÞW€
0040h: 5D CE 34 F7 42 1E 1C 75 96 EB D0 03 4F 41 E1 27  ]Î4÷B..u–ëÐ.OAá'
0050h: 63 04 E8 09 10 76 29 55 89 F1 D6 75 99 DA EA 87  c.è..v)U‰ñÖu™Úê‡
0060h: CE 3F DB B4 17 D2 00 25 87 98 1E CC 43 92 6A E8  Î?Û´.Ò.%‡˜.ÌC’jè
0070h: A4 4E 16 01 45 B8 D7 A5 83 FC CD DA CC F1 FA 20  ¤N..E¸×¥ƒüÍÚÌñú 
0080h: 26 FC 1D D2 F2 B2 34 C2 4D B9 FE 35 2A 6E DC 49  &ü.Òò²4ÂM¹þ5*nÜI
0090h: CA 67 AA B6 07 2B B2 8A 94 54 92 87 DB C0 91 73  Êgª¶.+²Š”T’‡ÛÀ‘s
00A0h: EC D6 2D 9B FA 02 F2 AA DE B5 71 51 1C 6B BA 6F  ìÖ-›ú.òªÞµqQ.kºo
00B0h: C4 D9 F9 94 90 6C 63 DB F4 29 F2 FD F0 7D 57 89  ÄÙù”.lcÛô)òýð}W‰
00C0h: E3 8D 57 B4 B0 79 65 C9 60 17 FB BD AA C0 9D AA  ã.W´°yeÉ`.û½ªÀ.ª
00D0h: F2 FC 82 47 CF DB E2 8E E9 39 8D B4 2B 7D EE 8D  òü‚GÏÛâŽé9.´+}î.
00E0h: 08 23 E6 D6 E2 B6 F9 59 A2 97 C5 10 64 2D A8 B6  .#æÖâ¶ùY¢—Å.d-¨¶
00F0h: B8 63 37 EB 17 81 47 0E 9F 3E D4 83 6C 45 69 46  ¸c7ë..G.Ÿ>ÔƒlEiF
0100h: E0 F0 43 19 A2 BF 91 2B BA 76 9C 32 46 53 38 FC  àðC.¢¿‘+ºvœ2FS8ü
0110h: D2 61 93 D7 25 58 99 7D E2 3E 80 44 05 F5 DB AF  Òa“×%X™}â>€D.õÛ¯
0120h: 0F 2A 17 16 43 1A 79 FB A8 7E 57 99 E8 E6 8F F3  .*..C.yû¨~W™èæ.ó
0130h: A0 39 99 08 71 3C 76 30 E4 4B 5F 8D 86 E9 3F C2   9™.q<v0äK_.†é?Â
0140h: 5E E1 8F 88 3B D5 18 3F 35 99 A9 E0 FD 36 AE 97  ^á.ˆ;Õ.?5™©àý6®—
0150h: 50 1B 85 C4 6E DB 83 93 D2 FD 4E 0E B8 29 1B 1A  P.…ÄnÛƒ“ÒýN.¸)..
0160h: F4 26 B0 25 CC 9A 6C 83 79 35 7A 8D 61 48 99 79  ô&°%Ìšlƒy5z.aH™y
0170h: 22 12 4E 1B 69 11 DE FE 29 FB 95 42 E8 DB CC 12  ".N.i.Þþ)û•BèÛÌ.
0180h: BC DE 56 78 F1 83 DB 1B 3A 2B 1D 66 4E A9 90 1A  ¼ÞVxñƒÛ.:+.fN©..
0190h: A0 67 15 E9 37 3A 17 BF F0 62 D6 2F DB 80 58 1C   g.é7:.¿ðbÖ/Û€X.
01A0h: EE 64 49 32 29 24 42 21 03 A9 B0 F6 E0 0B E8 2B  îdI2)$B!.©°öà.è+
01B0h: 42 C4 36 E3 33 47 29 17 73 DF C2 CB 48 01 FF 63  BÄ6ã3G).sßÂËH.ÿc
01C0h: 1F D1 06 A0 37 B6 38 83 9B 42 FA 09 1C 7D 28 05  .Ñ. 7¶8ƒ›Bú..}(.
01D0h: 52 BF 27 D2 0B F2 D1 FA 44 A3 EE CB C3 DD 73 D0  R¿'Ò.òÑúD£îËÃÝsÐ
01E0h: 2E A5 A5 4A 42 49 89 64 8F 95 CE F2 A3 76 FA 1F  .¥¥JBI‰d.•Îò£vú.
01F0h: 1A F7 E4 16 45 A1 98 52 1C 15 0D B3 86 F0 6C 7A  .÷ä.E¡˜R...³†ðlz
0200h: 8E 23 55 D8 66 5F C3 07 EC 3B AE 62 3F 25 13 02  Ž#UØf_Ã.ì;®b?%..
0210h: 96 D5 25 CF                                      –Õ%Ï
```

Fig 7: An example public RSA key present in one Conti sample

## THE FIRST "USABLE" SAMPLE

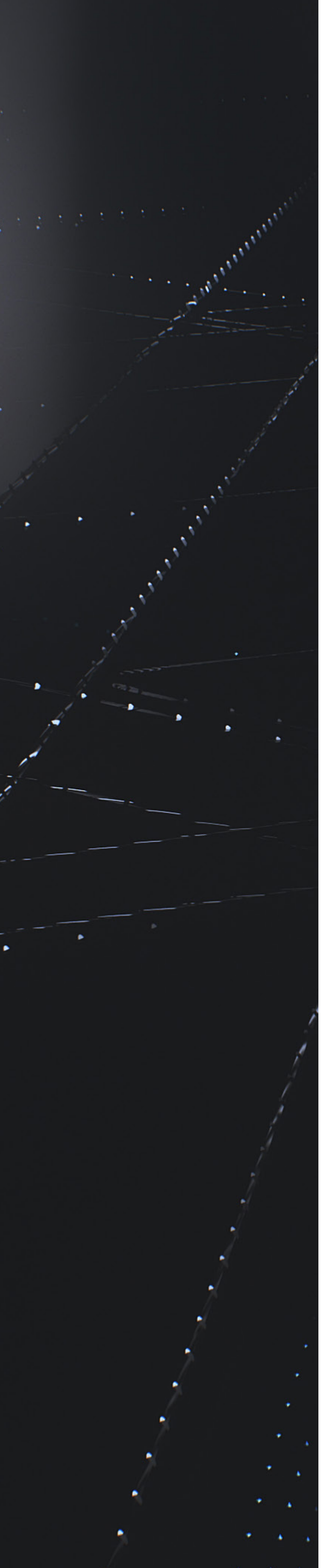
| Name | Hash (SHA-256) | Is Packed? | Timestamp (UTC) | Type |
|---|---|---|---|---|
| B1 | 308a561e0b874d7a356b916b2118288<br>a8c58d82a3ef26f136f6bdc45a388a692 | No | 2019/11/29 16:55:22 | EXE |
| B2 | 844cc2551f8bbfd505800bd3d135d930<br>64600a55c45894f89f80b81fea3b0fa1 | No | 2019/11/29 16:55:22 | EXE |

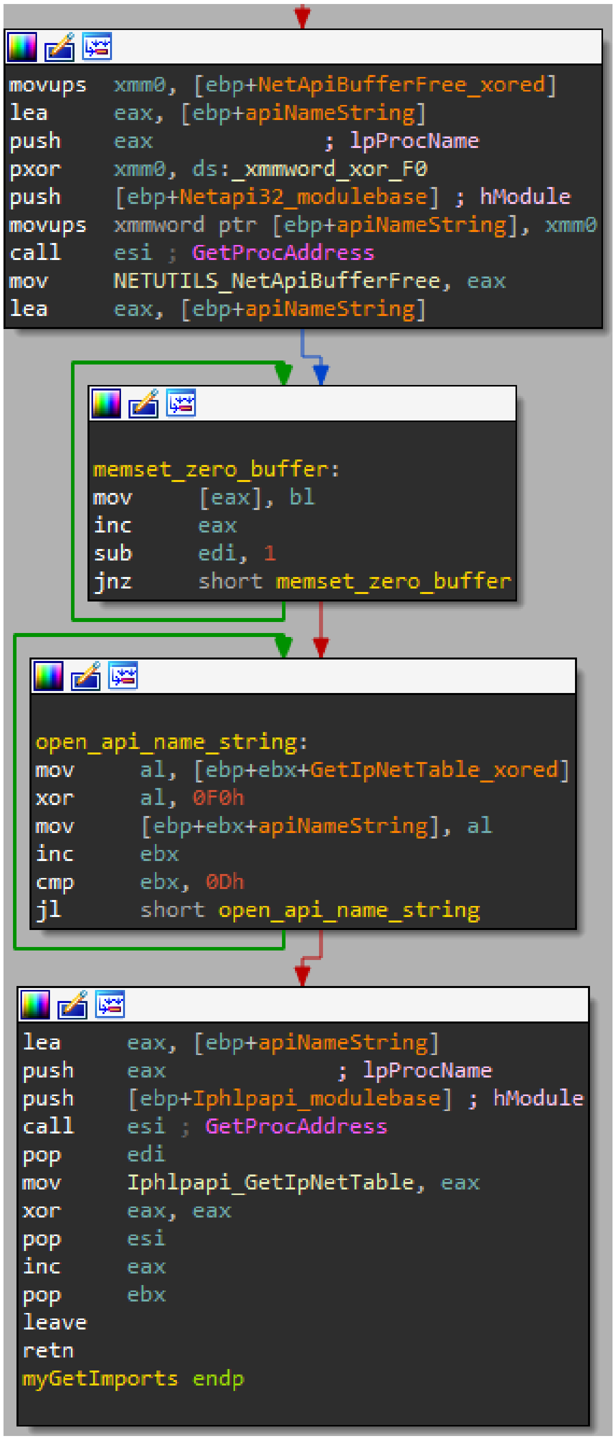
Two months later, we witness a newer version which went through a small amount of changes, one of which is the inclusion of a real ransom note instead of the "test note" previously embedded, allowing this sample to be used on a victim machine (hence, "usable", as otherwise the victim is left without a way to contact the attackers). From that we can assume this sample may have been tested on a machine of a real victim, although the first publicly known incident of Conti is dated to July 2020.

Looking at the import loading function, we observe a few things.

The first notable change is that the imports from Rstrtmgr.dll are no longer optional, and the function which loads APIs at runtime assumes they were found successfully, together with the other, less exotic imports. Because Rstrtmgr.dll is available on Windows Vista onward[4], we can infer this sample will crash on Windows XP when trying to unlock files. Considering the above, this function will always report success and the program will continue executing, never to reach the ExitProcess branch (which, somewhat ironically, is also loaded by this function now).

[4] https://docs.microsoft.com/en-us/windows/win32/rstmgr/restart-manager-portal





```
movups  xmm0, [ebp+NetApiBufferFree_xored]
lea     eax, [ebp+apiNameString]
push    eax             ; lpProcName
pxor    xmm0, ds:_xmmword_xor_F0
push    [ebp+Netapi32_modulebase] ; hModule
movups  xmmword ptr [ebp+apiNameString], xmm0
call    esi ; GetProcAddress
mov     NETUTILS_NetApiBufferFree, eax
lea     eax, [ebp+apiNameString]
```

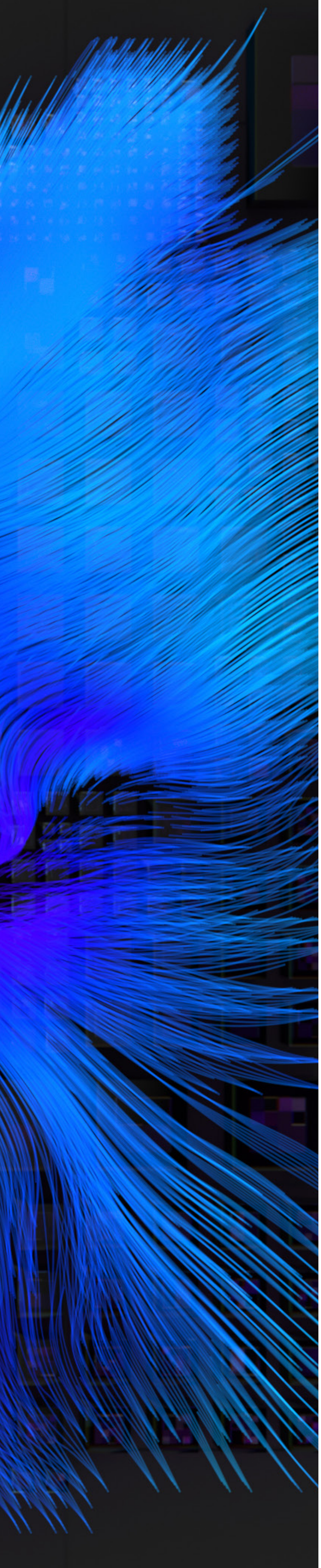
```
memset_zero_buffer:
mov     [eax], bl
inc     eax
sub     edi, 1
jnz     short memset_zero_buffer
```

```
open_api_name_string:
mov     al, [ebp+ebx+GetIpNetTable_xored]
xor     al, 0F0h
mov     [ebp+ebx+apiNameString], al
inc     ebx
cmp     ebx, 0Dh
jl      short open_api_name_string
```

```
lea     eax, [ebp+apiNameString]
push    eax             ; lpProcName
push    [ebp+Iphlpapi_modulebase] ; hModule
call    esi ; GetProcAddress
pop     edi
mov     Iphlpapi_GetIpNetTable, eax
xor     eax, eax
pop     esi
inc     eax
pop     ebx
leave
retn
myGetImports endp
```

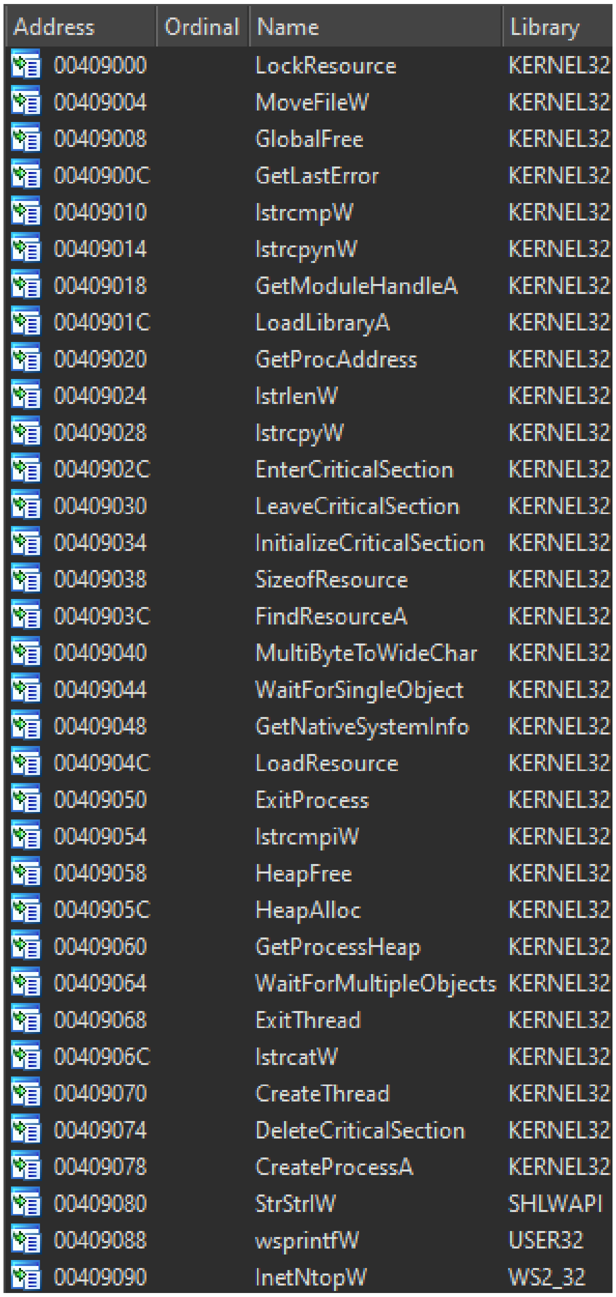
Fig 8: The imports loading function, always returning TRUE, without checking if the imports were found successfully.

The names of the APIs used in the import loading function are encoded with a different byte as a XOR key, 0x0F instead of 0x99. This shows a slight progression for making it harder to detect by static signatures and reversing.



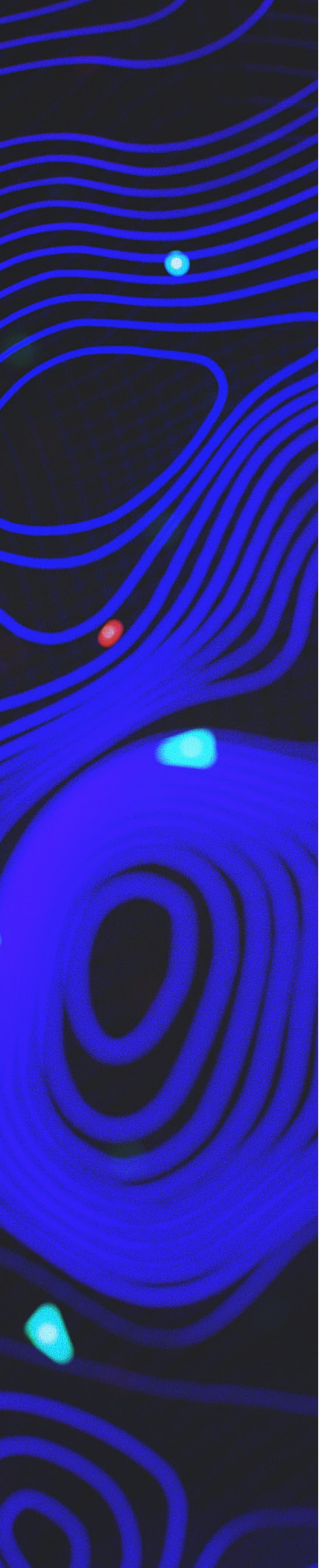
This time, all the imports are loaded at runtime instead of the weird mix that was in place previously, except for LoadLibraryA, GetProcAddress, for obvious reasons, which will be improved upon later. CreateThread is also in the load-time imports, for some odd reason, even though it is also loaded at runtime and used from both places. However, this sloppiness of having a mix of import origins will repeat itself later on.

Imports of sample A2:

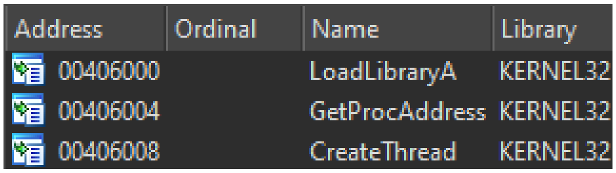


| Address | Ordinal | Name | Library |
|---|---|---|---|
| 00409000 | | LockResource | KERNEL32 |
| 00409004 | | MoveFileW | KERNEL32 |
| 00409008 | | GlobalFree | KERNEL32 |
| 0040900C | | GetLastError | KERNEL32 |
| 00409010 | | lstrcmpW | KERNEL32 |
| 00409014 | | lstrcpynW | KERNEL32 |
| 00409018 | | GetModuleHandleA | KERNEL32 |
| 0040901C | | LoadLibraryA | KERNEL32 |
| 00409020 | | GetProcAddress | KERNEL32 |
| 00409024 | | lstrlenW | KERNEL32 |
| 00409028 | | lstrcpyW | KERNEL32 |
| 0040902C | | EnterCriticalSection | KERNEL32 |
| 00409030 | | LeaveCriticalSection | KERNEL32 |
| 00409034 | | InitializeCriticalSection | KERNEL32 |
| 00409038 | | SizeofResource | KERNEL32 |
| 0040903C | | FindResourceA | KERNEL32 |
| 00409040 | | MultiByteToWideChar | KERNEL32 |
| 00409044 | | WaitForSingleObject | KERNEL32 |
| 00409048 | | GetNativeSystemInfo | KERNEL32 |
| 0040904C | | LoadResource | KERNEL32 |
| 00409050 | | ExitProcess | KERNEL32 |
| 00409054 | | lstrcmpiW | KERNEL32 |
| 00409058 | | HeapFree | KERNEL32 |
| 0040905C | | HeapAlloc | KERNEL32 |
| 00409060 | | GetProcessHeap | KERNEL32 |
| 00409064 | | WaitForMultipleObjects | KERNEL32 |
| 00409068 | | ExitThread | KERNEL32 |
| 0040906C | | lstrcatW | KERNEL32 |
| 00409070 | | CreateThread | KERNEL32 |
| 00409074 | | DeleteCriticalSection | KERNEL32 |
| 00409078 | | CreateProcessA | KERNEL32 |
| 00409080 | | StrStrIW | SHLWAPI |
| 00409088 | | wsprintfW | USER32 |
| 00409090 | | InetNtopW | WS2_32 |

Fig 9: Imports of Samples A2



Imports of sample B1:



| Address | Ordinal | Name | Library |
|---|---|---|---|
| 00406000 | | LoadLibraryA | KERNEL32 |
| 00406004 | | GetProcAddress | KERNEL32 |
| 00406008 | | CreateThread | KERNEL32 |

Fig 10: Imports of Samples B1

Instead of creating twice the amount of processes on the machine, it always uses 32 threads to read from the IoCompletionPorts. This simplification shouldn't improve encryption time on most systems, making the efficiency of it questionable.

The recursive iteration on drives, looking for files to encrypt, is parallelized. For each drive found, a new thread is created with the drive root path as parameter, and this thread just calls the iteration function from before, and exits after it completes. This is a nice addition that might boost speed. Also, this is the only place a load-time imported API is used (CreateThread).

Finally, there are some changes to the way encryption is performed:

- In previous versions, both the encrypted AES key and the original file size are written at the end of the file, before the encryption takes place. In this version, however, the original file size is written only after the file is fully encrypted. In our view, both methods should work pretty much the same.

- An additional small change is done to the way encrypted data is written back to the file. In this version, WriteFile is called in a loop for as long as the number of bytes written back to disk is not the amount intended to be written. This change might have been introduced to fix certain cases, where the encrypted bytes in memory did not match the actual bytes written to disk. This problem would cause the decryption to restore the wrong bytes.

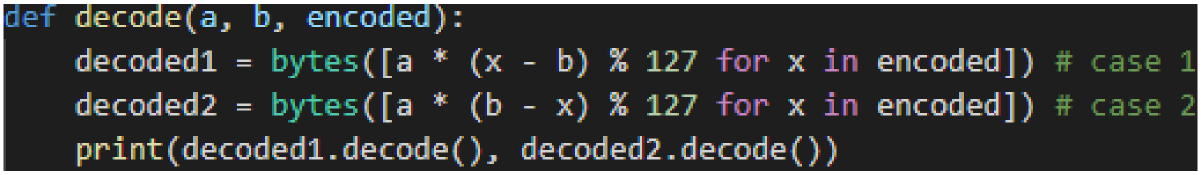
## MAKING SOME HEADLINES

| Name | Hash (SHA-256) | Is Packed? | Timestamp (UTC) | Type |
|---|---|---|---|---|
| C1 | eae876886f19ba384f55778634a35a1d<br>975414e83f22f6111e3e792f706301fe | No | 2020/06/04 00:02:10 | EXE |
| C2 | 1ef1ff8b1e81815d13bdd293554ddf8b3<br>e57490dd3ef4add7c2837ddc67f9c24 | No | 2020/06/04 00:02:10 | EXE |

We encounter the next iteration about half a year later. This was the first publicly analyzed Conti sample and it started to get headlines[5].

This iteration had the following additional features:

- String obfuscation got a considerable upgrade. Instead of a simple single-byte XOR, a custom encoding function is used, represented by the following pseudocode:



```
def decode(a, b, encoded):
    decoded1 = bytes([a * (x - b) % 127 for x in encoded]) # case 1
    decoded2 = bytes([a * (b - x) % 127 for x in encoded]) # case 2
    print(decoded1.decode(), decoded2.decode())
```

Fig 11: Improved string obfuscation method

  The constants (a, b) are different for every encoded string. Additionally, more strings are obfuscated in comparison with the previous samples, although some are still left open on the stack (i.e., DLL names).

- Two importing mechanisms are now shuffled together. Runtime APIs no longer require Rstrtmgr.dll APIs but now include net new imports. And yet additional APIs are resolved at load time. The disparity reinforces the view that multiple developers with different areas of responsibility may be involved in Conti.

[5] For a list of publications: https://malpedia.caad.fkie.fraunhofer.de/details/win.conti

- The ransomware will create a mutex and hold it, so that two instances won't execute at the same time, to avoid interfering with one another or simply slowing the encryption rate unnecessarily.

- Conti can now take command-line arguments. Those include:
  - An option to select the encryption mode (only local, only SMB shares, or both).
  - A list of network locations to search for shares and add files found on any such share to the encryption list.
  - These features allow the attackers to have a finer control on what Conti will encrypt without compiling something new.

- 36 fewer services are stopped[6], but the shadow copies and the termination of processes with "sql" in them is left unchanged.

- Only local (192.168.*; 10.*; and 172.*) IPs from the GetIpNetTable look for shares. This might explain the need for arguments to allow encrypting additional shares that might not adhere to any of the three specified IP patterns.

## MIXING IT UP

| Name | Hash (SHA-256) | Is Packed? | Timestamp (UTC) | Type |
|---|---|---|---|---|
| C1 | eae876886f19ba384f55778634a35a1d<br>975414e83f22f6111e3e792f706301fe | No | 2020/06/04 00:02:10 | EXE |
| C2 | 1ef1ff8b1e81815d13bdd293554ddf8b3<br>e57490dd3ef4add7c2837ddc67f9c24 | No | 2020/06/04 00:02:10 | EXE |

The next iteration of Conti appears around three months later and includes a greater number of changes, making it less similar to any previous version. Moreover, these changes make artifact-based clustering (e.g, by file extension of encrypted files, file structure, or ransom note content) less useful for determining whether a sample belongs to the Conti family, requiring deeper static or dynamic analysis to make the identification

Let's go over this long list of changes.

The entrypoint changed from a minimal C looking entry (which simply began with a call to the first function the malware author wrote, being the runtime import loading function) to a C++ bootstrap one, with all the jazz of CRT, security cookie, and SEH initializations. Later on, it's also apparent that unlike older versions, the C++ STL is being utilized– most notably, std::string is now used.

[6] See Appendix A for the changes.

The import loading went through a complete rework.

- For the most part, Conti does not embed the plain names of DLLs and their required exports, but instead, only keeps a hash of the strings it needs. To get the requisite imports, it iterates through NtCurrentPeb()->Ldr->InLoadOrderModuleList, at first looking for the module kernel32.dll by the hash of its name, later on finding the LoadLibraryA API in the same manner, iterating over exports until the hashes match.

- Only kernel32.dll is found by hash. The rest of the DLL names are embedded in the executable, now obfuscated, and are loaded using the LoadLibraryA API. An anecdote on that is, that even though kernel32 is loaded by its hash, in this early version, its name is still present in the binary and it is even decoded, even though it is never used.

- The API hashing[7] function of choice is Murmur2A[8] with a constant seed set to 0x5B2D, used on lowercase ASCII strings.

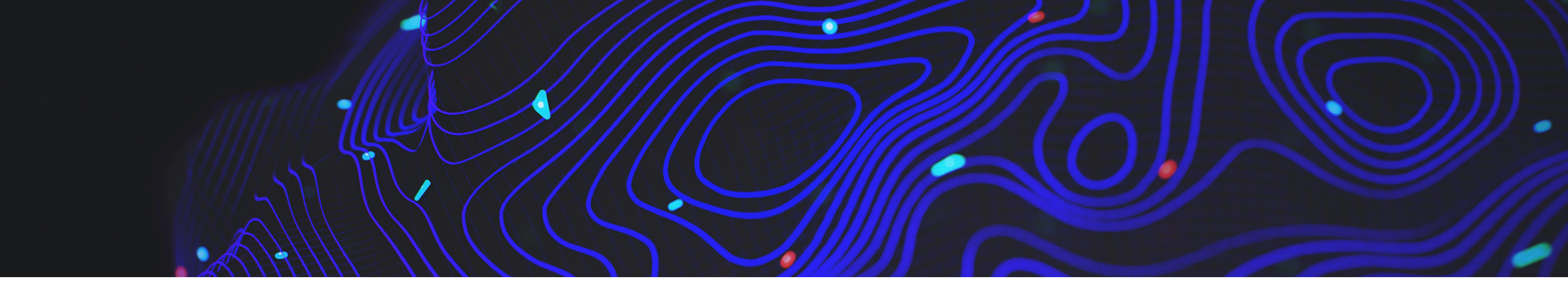
- A newly implemented hook removal logic takes place after loading all the necessary DLLs. For each loaded DLL, Conti reads its file on disk and goes through all the exports in it, looking for a difference in the first few bytes. If any such difference is found between the disk version and the in memory version, the bytes in memory are replaced by the bytes read from disk. Security products will often hook processes in order to fully monitor malicious activity. Conti targets this methodology specifically in the hopes of disarming security products lacking robust anti-tamper features.

- The APIs used in the hook removing function are acquired via the load-time linked LoadLibraryA and GetProcAddress, which once again is a new addition that didn't go through the refactor of switching APIs to those loaded at runtime.

Instead of fetching the needed imports at the beginning, each import is found by hash just before it is needed to be called. Moreover, to save relooking for previously fetched imports, they are cached in a single big array for sequential calls.

[7] Additional reading: https://www.ired.team/offensive-security/defense-evasion/windows-api-hashing-in-malware

[8] Implementation: https://github.com/rurban/smhasher/blob/4db9ed2dc7/MurmurHash2.cpp#L210-L247



With all those shenanigans accounted for, we find additional changes to the main logic:

- FAdded support for additional command-line arguments. In this version, a list of directories can be specified as targets for a search for interesting files to encrypt. Also, a flag for logging can be given, although it isn't implemented until a later version.

- The two resources have been removed. The ransom note content was moved to the data section of the executable, while the ability to encrypt specific file patterns instead of the default ones was removed. We speculate that the authors found the latter feature not worth supporting as we had never seen it used in practise.

- The ability to kill services was removed.

- The feature that allows deleting shadow copies is implemented differently, further down in the logic of this sample. In cases where the local encryption mode is chosen, the sample will delete shadow copies found from querying Win32_ShadowCopy on ROOT\CIMV2, by running a command using WMIC[9]. This is compared to previously using vssadmin[10] to delete a fixed amount of drives, which may or may not have shadow copies enabled, might not even be mounted on disk, or even miss a drive with shadow copy[11].

- Reimplementing a feature that was removed, the amount of threads created to read from the IoCompletionPorts is back to being based on the amount of processors available on the system. This time, it depends on what mode the ransomware is executed in, creating a thread per processor for cases where 'only local' or 'only network' encryption is specified, and creating twice as many if both modes are engaged. Interestingly, the reimplementation in this version is using the wrong value returned from the GetNativeSystemInfo query, using dwActiveProcessorMask instead of dwNumberOfProcessors. This small bug is fixed in a later version.

- This version includes changes to the implementation used to select files for encryption.

[9] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/wmic

[10] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/vssadmin

[11] See Appendix A for changes in commands

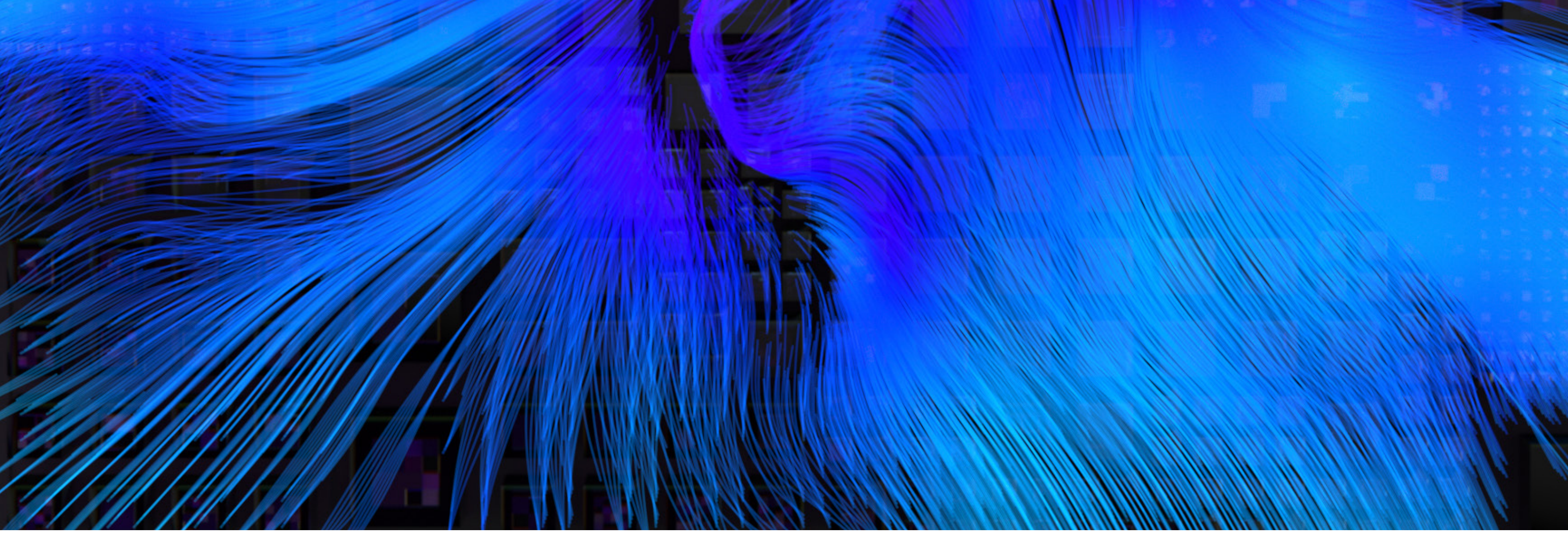
- This version includes changes to the implementation used to select files for encryption.
    - Different files are encrypted by different logic. We have a new list of 171 extensions where the whole content of the file is encrypted, and then, there is another list of 20 extensions for which only some parts of the file are encrypted. Lastly, the rest of the files are categorized by their size.
        - Files smaller than 1MiB are encrypted whole (same as the first extension list, making that list of probably known small files).
        - Files bigger than 1MiB and smaller than 5MiB have only their first 1MiB encrypted.
        - Files bigger than 5MiB are partially encrypted in jumps (same as the second extensions list, making that list of probably known big files)
    - The encryption algorithm is changed from AES to ChaCha. The keys are still generated randomly per file, and written to the end of the file after being encrypted with an embedded RSA public key located in the data section of the binary.
    - The extension of encrypted files were changed, from .CONTI to .YZXXX, which might help avoid detection of ransomware based on known extension changes.
- A significant change here is the implementation used to look for network shares to encrypt:
    - Addresses of the form “169.*” were added to the list of IP addresses to search for shares (along with “172.*”, “192.168.*”, and “10.*”, as mentioned above).
    - Instead of using InetNtopW to translate addresses to wide strings, now the inet_ntoa API is used, making the string only byte sized.
    - Instead of executing 32 threads to go search for shares on those IP addresses, only one thread is doing the dirty work.
    - There’s an additional check in the process of looking for network shares. Addresses are first checked to have an open port on 445, using sockets manually created by the malware. Then, NetShareEnum API is only called for IPs that answer the check. This change should reduce the noise of querying for shares on addresses that don’t have any.

## ADDITIONAL TOUCHES

| Name | Hash (SHA-256) | Is Packed? | Timestamp (UTC) | Type |
|---|---|---|---|---|
| E1 | 61dd6a0b2870d62f56c7fe0039d42bf5351588f927267fe7b4ee0761872a3b20 | No | 2020/09/23 21:17:34 | EXE |
| E2a | 9826b386065f8312a7a7ef431c735a66e85a9c144692907f5909f81f837c65f4 | Yes | 2020/10/21 18:39:10 | DLL |
| E3b | 96812dc56ffc07d9260b60dadd9729b1aaf29d426970f9385fad87d99d4578c2 | No | 2020/10/09 20:53:57 | EXE |
| E4 | 64a3a3ec70d20636299b8fe4f50c2b4d077f9934ee2d6ccf7d440b05b9770f56 | No | 2020/11/12 18:33:30 | EXE |
| E5 | 73bd8c2aa71f5dcd9d2ddd79e53656c6ae3db2535e08cf9dab1cd13bdd6d5ea3 | No | 2020/11/27 15:02:42 | EXE |
| E6 | 2fc6d7df9252b1e2c4eb3ad7d0d29c188d87548127c44cebc40db9abe8e5aa35 | No | 2020/12/23 16:26:10 | EXE |
| F1 | 5c5d05c4dcc9489ed527a1a607f0e2884d10558451662bcc849e36da7eca570c | No | 2021/01/12 19:20:18 | EXE |
| F2 | 3995502a85cc12c6962740989c4fb800d514bdf2ec667fdb7e4c8206adca0235 | No | 2021/02/04 15:57:28 | EXE |
| G1 | 4478feb1e3c98220f50ce341665087b7f6c1d9c290e42f54812bc55da5b3707d | No | 2021/03/08 14:33:55 | EXE |
| G2 | d29b8160e51dd29474f3464111fc888da8adb2bc2f0d4f29ce71219ffc846bd5 | No | 2021/04/20 20:53:59 | EXE |

From this point on, we find samples more frequently, both packed and unpacked. Some samples are practically the same, except for the embedded public RSA key, the extension used for encrypted files, and the text placed inside the "readme" text file. A good post which goes over an E sample thoroughly, including all the features we discussed so far, can be found here[12]. Other than that, most changes going forward per new sample are minor, and therefore they are only aggregated in the Summary table below.

[12] http://chuongdong.com/reverse%20engineering/2020/12/15/ContiRansomware/

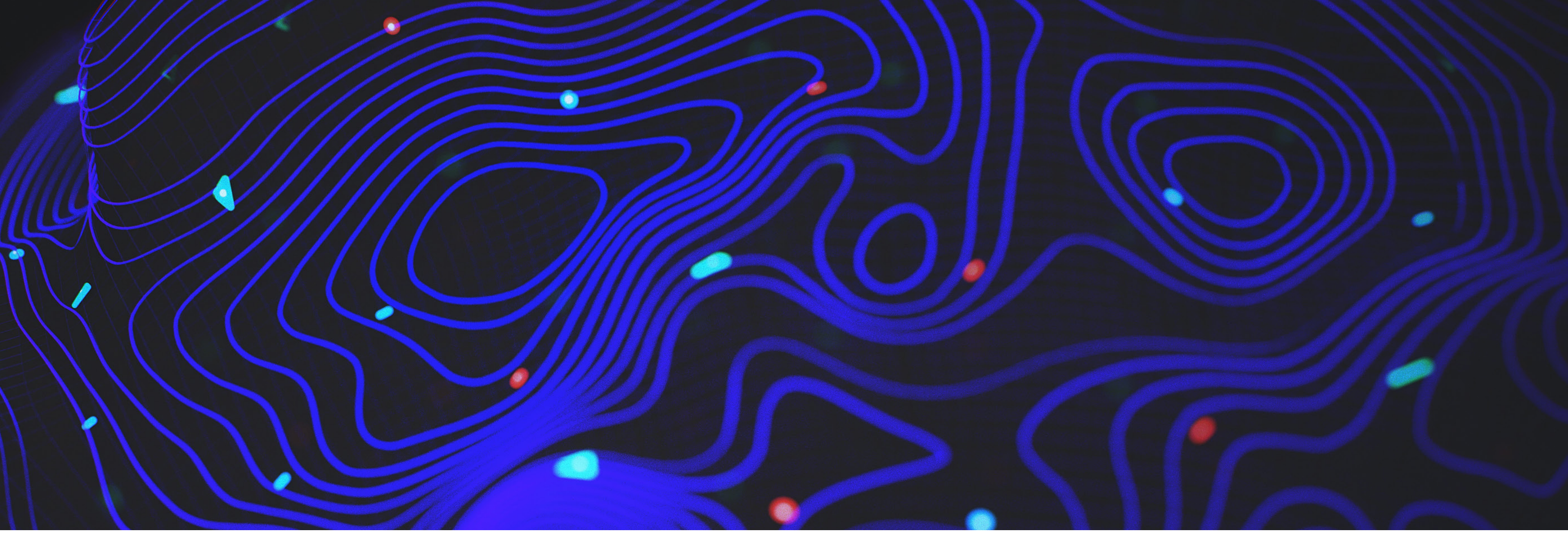


## SUMMARY

Below is a summarized list of all the major Conti milestones:

| Variant | Version/Date | Features/Updates |
|---|---|---|
| A1c, A1b, A1c, A2 | 2019/10/06 | **Imports:** Conti has only a few of its imported functions linked at load-time. Moreover, all the names of the APIs are encoded using a simple XOR with the byte 0x99. |
| B1, B2 | 2019/11/29 | **Imports:** The names of the APIs are encoded with a different XOR key, 0x0F instead of 0x99; All the imports are loaded at runtime;<br>**Multithreading:** Now it uses 32 threads for file encryption; the recursive iteration on drives for encryption got parallelized as well.<br>**File encryption:** Small stability improvement to the writing phase of file encryption |
| C1, C2 | 2020/06/04 | **String obfuscation improvement:** instead of a simple XOR with a single byte, a custom encoding function is used.<br>**Mutex creation:** The ransomware now create a mutex on system to avoid concurrent infections;<br>This version implements **command line arguments** including an option to select the encryption mode and a list of network locations to search for.<br>**Minor:** 36 less services are stopped |

| Variant | Version/Date | Features/Updates |
|---|---|---|
| D1a, D1b, D1c | 2020/09/02 | **Imports:** introduction of API hashing using the function Murmur 2A with a constant seed set to 0x5B2D. To save relooking for imports, they are cached in a single big array for sequential calls<br>**New command line arguments:** a flag for logging and an option to specify specific directories is introduced.<br>**The ransom note** is moved from the .rsrs section to the .data section.<br>**Deletion of shadow copies** with wmic instead of vssadmin.<br>**Multithreading:** Now, the amount of allocated thread for encryption is based on the amount of processors available on the systems. The exact number of the thread depends on what mode the ransomware is executed in.<br>**File encryption:** A new and different logic for file encryption is implemented. 2 modes for encrypting files: full and partial, depending on file extension and file dimension.<br>Instead of executing 32 threads to go search for shares on those IP addresses, only one thread does so.<br>Check for an open port before querying for shares on remote IPs. |
| E1, E2a, E3b, E4, E5, E6 | 2020/10 - 2020/12 | **Ransom note:** contains more contact information (website, TOR node and email) and a UUID.<br>**Logging option:** The new command line option is now implemented for logging errors.<br>**Single directory** encryption mode instead of traversal.<br>Dead code & busy loops to hinder simulation / static analysis. |
| F1, F2 | 2021/01 - 2021/02 | **The mutex** creation is now optional from the command line.<br>**Optimization of file encryption:** encryption through IoCompletionPorts was replaced by C++ queues and locks.<br>The seed of the Murmur hash function changed to 0xB801FCDA. |
| G1, G2 | 2021/03 - 2021/04 | PathIsDirectoryW API used to sepwarate files from directories.<br>The seed of the Murmur hash function changed to 0xFF889912. |



## CONCLUSION

We took a deep dive into the evolution of Conti ransomware, gaining some insight into the process of developing ransomware. Most notably, we saw how many changes take place to increase the evasiveness of the malware from detections and complicate the analysis process. Most meaningful changes and additions to the ransomware were done prior to September-October 2020, at which point, the developers needed only to make minor refinements to stay ahead of detections and keep the money rolling in for its affiliates. Today, Conti is a mature project that is being used actively and aggressively to compromise and extort victims on a daily basis

## APPENDIX A

### Shadow Copies Deletion

#### Deletion of Shadow Copies (vssadmin):

```
cmd.exe /c vssadmin Delete Shadows /all /quiet
cmd.exe /c vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB
cmd.exe /c vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded
cmd.exe /c vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB
cmd.exe /c vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded
cmd.exe /c vssadmin resize shadowstorage /for=e: /on=e: /maxsize=401MB
cmd.exe /c vssadmin resize shadowstorage /for=e: /on=e: /maxsize=unbounded
cmd.exe /c vssadmin resize shadowstorage /for=f: /on=f: /maxsize=401MB
cmd.exe /c vssadmin resize shadowstorage /for=f: /on=f: /maxsize=unbounded
cmd.exe /c vssadmin resize shadowstorage /for=g: /on=g: /maxsize=401MB
cmd.exe /c vssadmin resize shadowstorage /for=g: /on=g: /maxsize=unbounded
cmd.exe /c vssadmin resize shadowstorage /for=h: /on=h: /maxsize=401MB
cmd.exe /c vssadmin resize shadowstorage /for=h: /on=h: /maxsize=unbounded
cmd.exe /c vssadmin Delete Shadows /all /quiet
```

#### Deletion of Shadow Copies template (WMI, replacing %s with a shadow copy ID):

```
cmd.exe /c C:\Windows\System32\wbem\WMIC.exe shadowcopy where “ID=’%s’” delete
```

### Services Stopping

#### Command template for service stopping (replacing %s with a service name):

```
cmd.exe /c net stop %s /y
```

Stopped services list
(extracted from sample A2 in this document; sorted):

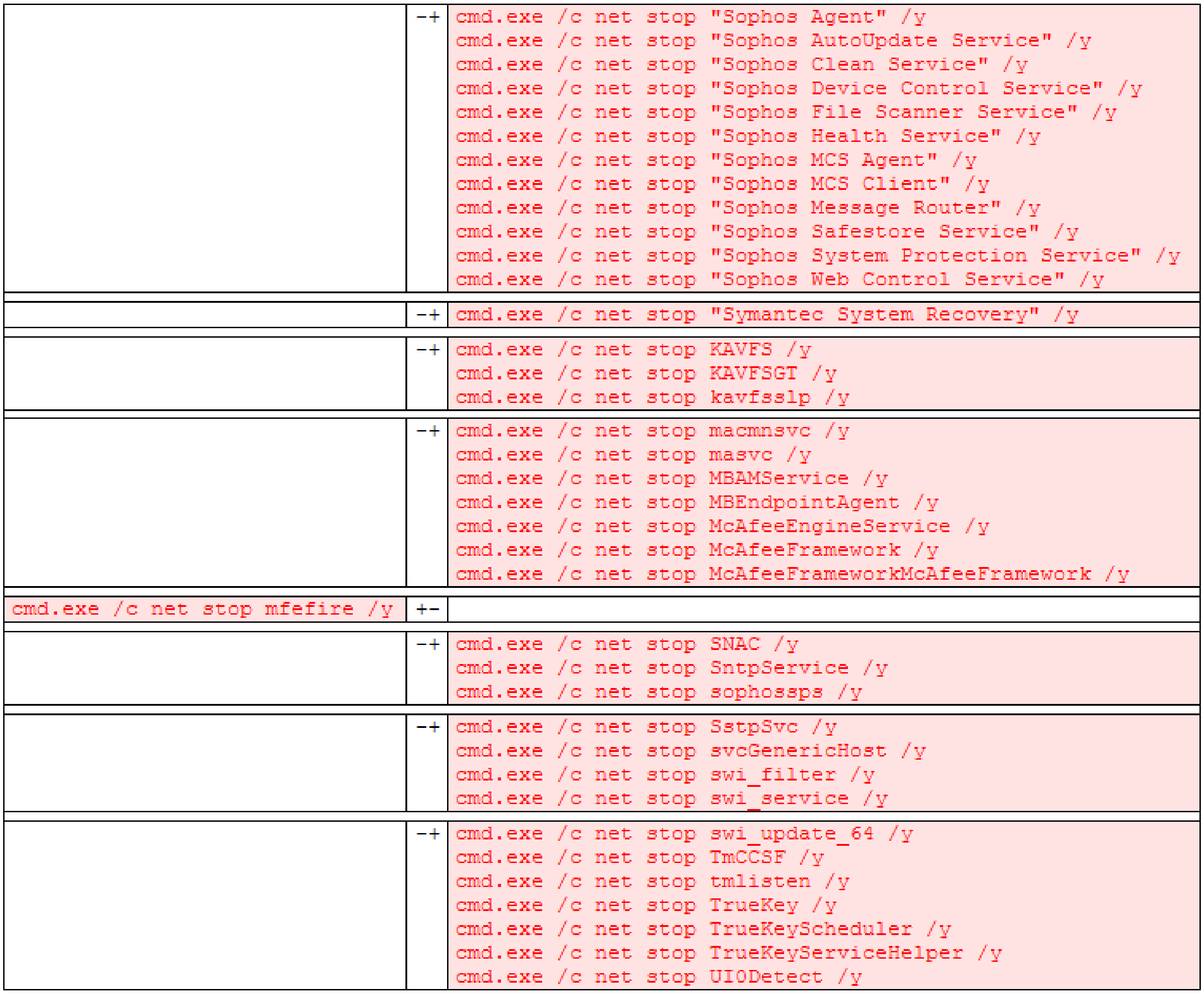
```
“Acronis VSS Provider”
“Enterprise Client Service”
“Sophos Agent”
“Sophos AutoUpdate Service”
“Sophos Clean Service”
“Sophos Device Control Service”
“Sophos File Scanner Service”
“Sophos Health Service”
“Sophos MCS Agent”
“Sophos MCS Client”
“Sophos Message Router”
“Sophos Safestore Service”
“Sophos System Protection Service”
“Sophos Web Control Service”
“SQL Backups”
“SQLsafe Backup Service”
“SQLsafe Filter Service”
“Symantec System Recovery”
“Veeam Backup Catalog Data Service”
“Zoolz 2 Service”
AcronisAgent
AcrSch2Svc
Antivirus
ARSM
AVP
BackupExecAgentAccelerator
BackupExecAgentBrowser
BackupExecDeviceMediaService
BackupExecJobEngine
BackupExecManagementService
BackupExecRPCService
BackupExecVSSProvider
bedbg
DCAgent
EhttpSrv
ekrn
EPSecurityService
EPUpdateService
EraserSvc11710
EsgShKernel
ESHASRV
FA_Scheduler
IISAdmin
IMAP4Svc
KAVFS
KAVFSGT
```

```
kavfsslp
klnagent
macmnsvc
masvc
MBAMService
MBEndpointAgent
McAfeeEngineService
McAfeeFramework
McAfeeFrameworkMcAfeeFramework
McShield
McTaskManager
mfemms
mfevtp
MMS
mozyprobackup
MsDtsServer
MsDtsServer100
MsDtsServer110
MSExchangeES
MSExchangeIS
MSExchangeMGMT
MSExchangeMTA
MSExchangeSA
MSExchangeSRS
msftesql$PROD
MSOLAP$SQL_2008
MSOLAP$SYSTEM_BGC
MSOLAP$TPS
MSOLAP$TPSAMA
MSSQL$BKUPEXEC
MSSQL$ECWDB2
MSSQL$PRACTICEMGT
MSSQL$PRACTTICEBGC
MSSQL$PROD
MSSQL$PROFXENGAGEMENT
MSSQL$SBSMONITORING
MSSQL$SHAREPOINT
MSSQL$SOPHOS
MSSQL$SQL_2008
MSSQL$SQLEXPRESS
MSSQL$SYSTEM_BGC
MSSQL$TPS
MSSQL$TPSAMA
MSSQL$VEEAMSQL2008R2
MSSQL$VEEAMSQL2008R2
MSSQL$VEEAMSQL2012
```

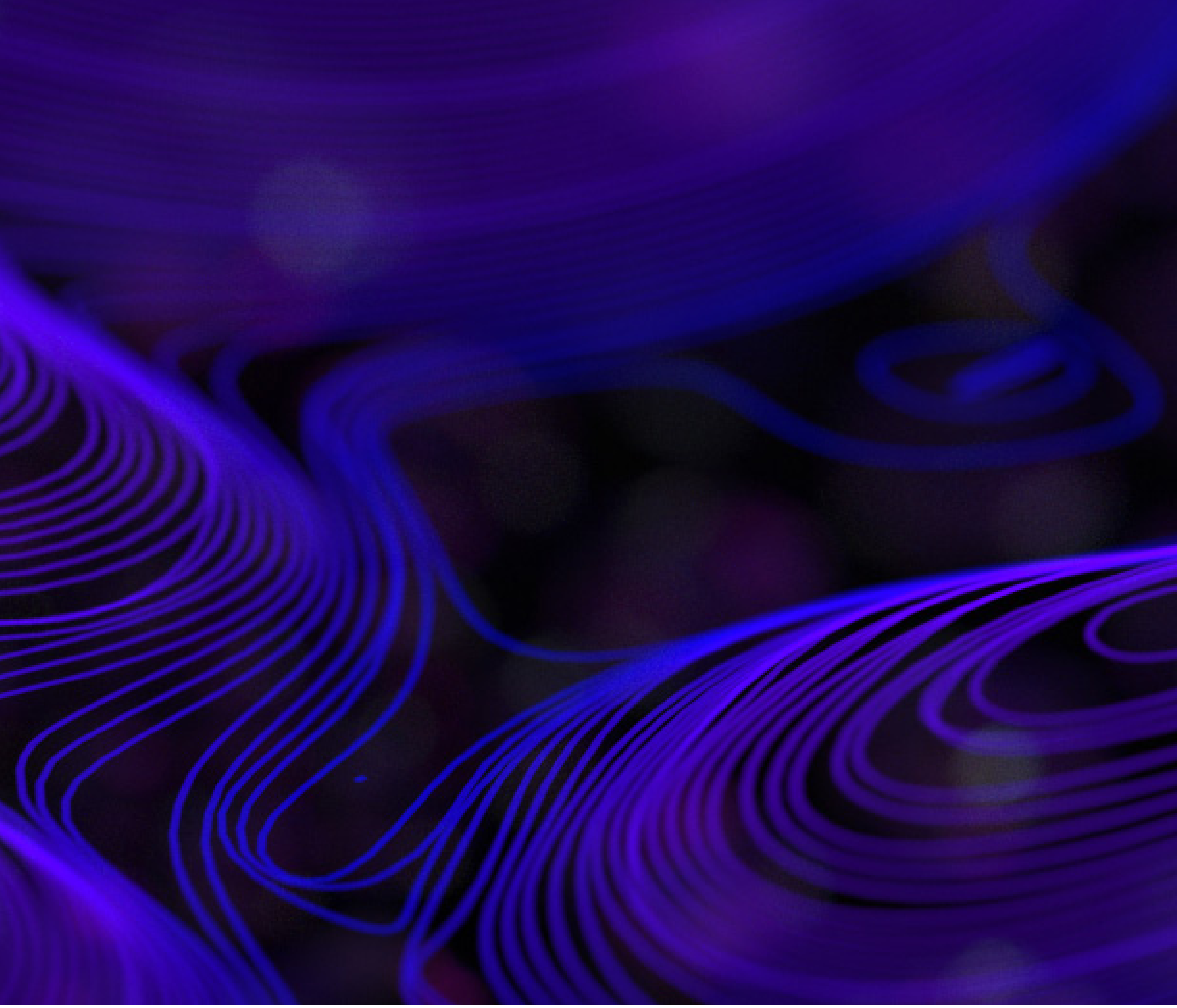
```
MSSQLFDLauncher
MSSQLFDLauncher$PROFXENGAGEMENT
MSSQLFDLauncher$SBSMONITORING
MSSQLFDLauncher$SHAREPOINT
MSSQLFDLauncher$SQL_2008
MSSQLFDLauncher$SYSTEM_BGC
MSSQLFDLauncher$TPS
MSSQLFDLauncher$TPSAMA
MSSQLSERVER
MSSQLServerADHelper
MSSQLServerADHelper100
MSSQLServerOLAPService
MySQL57
NetMsmqActivator
ntrtscan
OracleClientCache80
PDVFSService
POP3Svc
ReportServer
ReportServer$SQL_2008
ReportServer$SYSTEM_BGC
ReportServer$TPS
ReportServer$TPSAMA
RESvc
sacsvr
SamSs
SAVAdminService
SAVService
SDRSVC
SepMasterService
ShMonitor
Smcinst
SmcService
SMTPSvc
SNAC
SntpService
sophossps
SQLAgent$BKUPEXEC
SQLAgent$CITRIX_METAFRAME
SQLAgent$CXDB
SQLAgent$ECWDB2
SQLAgent$PRACTTICEBGC
SQLAgent$PRACTTICEMGT
SQLAgent$PROD
SQLAgent$PROFXENGAGEMENT
SQLAgent$SBSMONITORING
```

```
SQLAgent$SHAREPOINT
SQLAgent$SOPHOS
SQLAgent$SQL_2008
SQLAgent$SQLEXPRESS
SQLAgent$SYSTEM_BGC
SQLAgent$TPS
SQLAgent$TPSAMA
SQLAgent$VEEAMSQL2008R2
SQLAgent$VEEAMSQL2008R2
SQLAgent$VEEAMSQL2012
SQLBrowser
SQLSafeOLRService
SQLSERVERAGENT
SQLTELEMETRY
SQLTELEMETRY$ECWDB2
SQLWriter
SstpSvc
svcGenericHost
swi_filter
swi_service
swi_update
swi_update_64
TmCCSF
tmlisten
TrueKey
TrueKeyScheduler
TrueKeyServiceHelper
UI0Detect
VeeamBackupSvc
VeeamBrokerSvc
VeeamCatalogSvc
VeeamCloudSvc
VeeamDeploymentService
VeeamDeploySvc
VeeamEnterpriseManagerSvc
VeeamHvIntegrationSvc
VeeamMountSvc
VeeamNFSSvc
VeeamRESTSvc
VeeamTransportSvc
W3Svc
wbengine
wbengine
WRSVC
```

Diff between services stopped on a later version (C1; left) and the earlier one (A2; right):



| | | |
|---|---|---|
| | -+ | cmd.exe /c net stop "Sophos Agent" /y<br>cmd.exe /c net stop "Sophos AutoUpdate Service" /y<br>cmd.exe /c net stop "Sophos Clean Service" /y<br>cmd.exe /c net stop "Sophos Device Control Service" /y<br>cmd.exe /c net stop "Sophos File Scanner Service" /y<br>cmd.exe /c net stop "Sophos Health Service" /y<br>cmd.exe /c net stop "Sophos MCS Agent" /y<br>cmd.exe /c net stop "Sophos MCS Client" /y<br>cmd.exe /c net stop "Sophos Message Router" /y<br>cmd.exe /c net stop "Sophos Safestore Service" /y<br>cmd.exe /c net stop "Sophos System Protection Service" /y<br>cmd.exe /c net stop "Sophos Web Control Service" /y |
| | | |
| | -+ | cmd.exe /c net stop "Symantec System Recovery" /y |
| | | |
| | -+ | cmd.exe /c net stop KAVFS /y<br>cmd.exe /c net stop KAVFSGT /y<br>cmd.exe /c net stop kavfsslp /y |
| | | |
| | -+ | cmd.exe /c net stop macmnsvc /y<br>cmd.exe /c net stop masvc /y<br>cmd.exe /c net stop MBAMService /y<br>cmd.exe /c net stop MBEndpointAgent /y<br>cmd.exe /c net stop McAfeeEngineService /y<br>cmd.exe /c net stop McAfeeFramework /y<br>cmd.exe /c net stop McAfeeFrameworkMcAfeeFramework /y |
| | | |
| cmd.exe /c net stop mfefire /y | +- | |
| | | |
| | -+ | cmd.exe /c net stop SNAC /y<br>cmd.exe /c net stop SntpService /y<br>cmd.exe /c net stop sophossps /y |
| | | |
| | -+ | cmd.exe /c net stop SstpSvc /y<br>cmd.exe /c net stop svcGenericHost /y<br>cmd.exe /c net stop swi_filter /y<br>cmd.exe /c net stop swi_service /y |
| | | |
| | -+ | cmd.exe /c net stop swi_update_64 /y<br>cmd.exe /c net stop TmCCSF /y<br>cmd.exe /c net stop tmlisten /y<br>cmd.exe /c net stop TrueKey /y<br>cmd.exe /c net stop TrueKeyScheduler /y<br>cmd.exe /c net stop TrueKeyServiceHelper /y<br>cmd.exe /c net stop UI0Detect /y |



# ABOUT SENTINELLABS

InfoSec works on a rapid iterative cycle where new discoveries occur daily and authoritative sources are easily drowned in the noise of partial information. SentinelLabs is an open venue for our threat researchers and vetted contributors to reliably share their latest findings with a wider community of defenders. No sales pitches, no nonsense. We are hunters, reversers, exploit developers, and tinkerers shedding light on the world of malware, exploits, APTs, and cybercrime across all platforms. SentinelLabs embodies our commitment to sharing openly –providing tools, context, and insights to strengthen our collective mission of a safer digital life for all. In addition to Microsoft operating systems, we also provide coverage and guidance on the evolving landscape that lives on Apple and macOS devices. https://labs.sentinelone.com/