CRIMEWARE

Conti Unpacked | Understanding Ransomware Development As a Response to Detection

IDAN WEIZMAN / JULY 8, 2021

By Idan Weizman & Antonio Pirozzi

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 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. In this post, we summarize our main findings.

Read the Full Report

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.

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 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:

Reported Conti incidents – 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.

Conti – Evolution With Focus

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. In particular, we see a focus on the following key ransomware characteristics across the evolution of Conti variants:

Obfuscation: 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, creating difficulties for static analysis and detection tools.

Speed: Conti uses up to 32 concurrent CPU threads for file encryption operations. Starting from the iteration of September 2020, the developers switched from AES to the CHACHA algorithm to further speed up the encryption process. This translates into less time required to lock victims' data and reduce the chance of the operation being blocked.

File Encryption: starting from 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. From January 2021, encryption through `IoCompletionPorts` was replaced by C++ queues and locks.

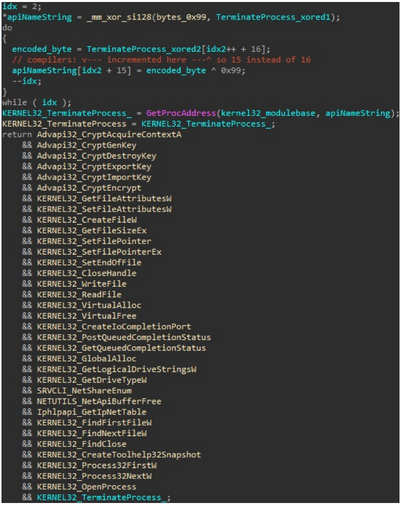
The Early Samples

The earliest sample of Conti we found dates from the end of 2019 and includes indications that it's an early test version (e.g., the ransom note contains the text "test note"). It took eight months for this version to make headlines, but analysis of this 'prototype' helps us understand how Conti developed over time.

```
SHA-256: 2f334c0802147aa0eee90ff0a2b0e1022325b5cba5cb5236ed3717a2b0582a9c
Packed: Yes
Timestamp: 2019/10/06 14:08:28
File Type: EXE

SHA-256: 4f43a66d96270773f4e849055a844feb6ef234d7340b797f8763b7a9f8d80583
Packed: Yes
Timestamp: 2019/10/06 12:43:23
File Type: DLL

SHA-256: 94bdec109405050d31c2748fe3db32a357f554a441e0eae0af015e8b6461553e
Packed: No
Timestamp: 2019/10/21 15:00:01
File Type: EXE

SHA-256: 77b1fcae9e8f0a5a739c35961382e2b3f239a05c1135c4a8efe1964a263d5a47
Packed: No
Timestamp: 2019/10/21 15:00:01
File Type: EXE
```

These early samples have only a few imported functions linked at load time. Therefore, the first thing the code does is manually load required libraries at runtime using `LoadLibraryA` and `GetProcAddress`.

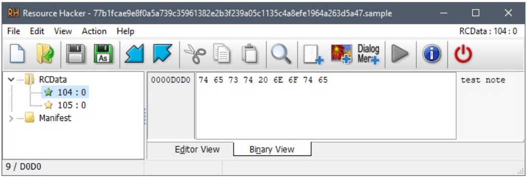
Moreover, all API names are encoded using a simple XOR with the byte 0x99. The names of DLLs are not encoded in this early version, save for some optional imports from `Rstrtmgr.dll`, the DLL responsible for Microsoft's Windows' Restart Manager function. The `GetProcAddress` function ends by making sure 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_;
```

Getting the last import and checking all imports are found

Two resources loaded from the PE file are of particular note. 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 comma-separated strings denoting files that should be encrypted in case they contain a substring from the list.

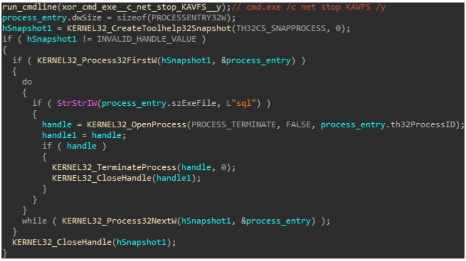


```
0000D0D0  74 65 73 74 20 6E 6F 74 65    test note
```

The hardcoded ransom note

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.

In this early version, 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);
}
```

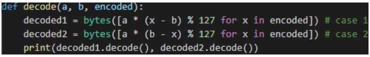
Terminating processes containing ‘sql’

Our full technical report explores more details of this prototype version, but the last point we shall note here is that at the end of the encryption process, the file will be moved, adding the extension `.CONTI` to the end of it.

Conti Appears In The Wild

Two months later a new version appeared with the inclusion of a real ransom note instead of the embedded “test note”. Other minor changes include changes to the XOR key from 0x99 to 0x0F. More significantly, the ransomware now loads all imports at runtime, with the exception of `LoadLibraryA`, `GetProcAddress`, and for some unknown reason, `CreateThread`. This import is used to boost speed through parellelization as the ransomware looks for files to encrypt across all available drives.

Six months later, in July 2020, Conti had a third iteration and hit the headlines for the first time. String obfuscation has received a significant upgrade with the single-byte XOR key replaced by a custom encoding function, represented by the following pseudo code:



```
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())
```

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).

There are further changes to how APIs are loaded, but a noticeable lack of consistency, which reinforces the view that multiple developers with different areas of responsibility may be involved in Conti.

A notable new feature is the ability to accept command line arguments, meaning Conti can now be controlled by a human operator for improved targeting. The options include the ability to select the encryption mode (only local, only SMB shares, or both) as well as allowing a list of network locations to search for shares, and adding files found on such shares to the encryption list.

Conti’s Developers Respond To Detection Engines

By September 2020, Conti was making bigger waves, with press reports of an attack on the Fourth District Court of Louisiana claiming the U.S. court’s website was knocked offline and that stolen documents relating to defendants, witnesses and jurors were leaked.

By this time, Conti was on the radar of most endpoint security solutions and the developers clearly took notice. The next iteration includes a greater number of changes than the previous versions, with a heavy emphasis on evasion and anti-analysis.

For the most part, Conti now 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.

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. This feature is aimed at bypassing some modern EPP/EDR platforms. 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.

There are a number of significant changes to the main logic, features and encryption, explored in greater detail in the full technical report. For example, 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.

Ever-focused on speed to beat mitigation attempts, Conti now includes a hardcoded list of 171 file extensions for which the whole content of the file is encrypted along with a further list of 20 file extensions for which only some part of the file is encrypted. Other files are categorized by size such that:

- Files smaller than 1MiB are encrypted whole.
- Files larger than 1MiB and smaller than 5MiB have only their first 1MiB encrypted.
- Files larger than 5MiB are partially encrypted in jumps.

The extension of encrypted files is now changed from `.CONTI` to `.YZXXX` in a bid to avoid simple ransomware detection logic based on known extension changes.

Refining a Successful RaaS Model

Late 2020 saw further iteration with Conti now refining its ransom note to contain more contact information including website, TOR node, email and a “customer” UUID.

Example of recent Conti ransom note

Affiliates were offered a new command line option for logging errors as well as other improvements. To keep detection engines at bay, Conti included more dead code and busy loops to hinder simulation and static analysis.

Through early 2021, the developers changed the seed for their custom hash function twice across two more iterations. 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 file. Other than that, most changes going forward per new sample are minor.

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 the detection curve and keep the money rolling in for their affiliates. Today, Conti is a mature project that is being used actively and aggressively to compromise and extort victims on a daily basis. Read the full report for further details and a complete list of IOCs.

Read the Full Report

Read the Full Report

RAAS RANSOMWARE

IDAN WEIZMAN

Idan Weizman has been a security researcher at SentinelOne since 2017, following four years service in the IDF. As a researcher, Idan focuses on reverse engineering, Windows internals and malware analysis. Outside of work, he enjoys gaming, watching anime and solving problems in mathematics.

©2023 SentinelOne, All Rights Reserved.