Ransomware data recovery techniques

Irimia Alexandru-Vasile

Faculty of Computer Science, Alexandru Ioan Cuza University of Iasi,

Email: alexviirimia@gmail.com

Abstract

This article presents and explains methodologies that can be employed to recover information from encrypted files generated by ransomware based on cryptanalytic techniques. By using cryptanalysis and related knowledge as much as possible, the methodology's goal is to use static and dynamic analysis as little as possible. We present three case studies that illustrate different approaches that can be used to recover the encrypted data.

Keywords: cryptanalysis; ransomware; stolen information; encrypted data

1. Introduction

Nowadays, malware attacks are becoming more and more frequent to disrupt services, steal sensitive information, block access to data and much more. In this article we will be talking about ransomware, a certain type of malware that threatens its victims to destroy or block access to data or systems until a ransom is paid.

Many companies, as well as individuals, are affected annually by this type of malware. According to [1], in 2019, a new company was infected with ransomware every 14 seconds. The cost of these infections is also high, as a ransom pay for a small business is on average $5,900 (see [3]), but it can be as high as $50 million, the highest demand in history that Acer was requested in 2021 (see [4]). Most of the time it is less costly to not pay the ransom, as seen in [2], in 2020 the average ransom pay was $1,450,000, while the cost to recover from the attack without paying the ransom was $732,000. Even

by paying the ransom, victims do not get all their data back. As seen in [2], files get corrupted in the decryption process and only recover around 65 percent of data due to technical faults in the ransomware itself. On the bright side, businesses affected by ransomware retrieved their data in 57 percent of cases utilizing backups. A further 8 percent of people recovered their data using other methods. This results in a 97 percent data recovery rate when combined with the 32 percent who actually paid the ransom (see [2]).

Ransomwares take various form, the most common being:

- **Crypto ransomware or encryptors** are the most well-known and the most damaging. They encrypt files and data within a system, then ask the victim to pay a ransom to recover them.
- **Lockers** block computer functions such as being able to use your mouse and keyboard, or not being able to access the desktop, making the computer inoperable until the ransom demands are met.
- **Scareware** is fake software that claims to have detected an issue on the victim's computer or a virus and demands payment to solve those problems. Some of these malwares lock the computer or flood the screen with pop-up alerts.
- **Doxware\Leakware** threatens to distribute sensitive or personal information online**.**
- **Raas (Ransomware as a service)** is a variant that is anonymously hosted by a "professional" hacker that handles all aspects of the attack in return for a cut of the ransom.

In this article we will be talking about **crypto ransomware** and methodologies used to recover the encrypted data based on cryptanalytic and reverse engineering techniques.

2. History of ransomware

The first ransomware was **AIDS Trojan** (see [5])**,** also known as **PC Cyborg,** created in 1989 by Joseph Popp and was distributed to 20,000 attendees at the 1989 World Health Organization AIDS conference, hence the name. The malware was distributed over a floppy disk, so it relied only on researchers' interest about what was on the disk, not on any kind of exploit. There was a questionnaire regarding AIDS on the floppy disk. The application was installed by researchers, and everything worked as intended on their computers up to the 90th restart. The ransomware would encrypt the victim's filenames, but not the contents of the files. It would also demand $189 in licensing fees for the PC Cyborg Software, which were to be paid by cashier's check or international money order and sent to a Panama P.O. Box, but later a decryptor called **CLEARAID** was developed that would restore the files without paying the ransom.

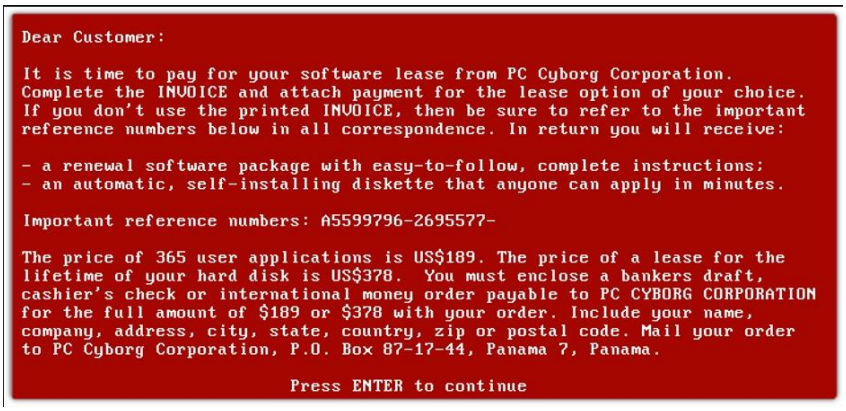

Figure 1. **AIDS Trojan** ransom note

More than a decade later, in 2004, wave of ransomware infections starts taking place, beginning with **GPCoder** identified by Symantec as a Trojan that in its September 2005 Internet Security Threat Report "encrypts data files such as documents, spreadsheets, and database files on the compromised computer," although it was not labeled as ransomware. A note demanding $200 as ransom would be left in each directory. The next year, the **Archiveus Trojan** tried a slightly different strategy in 2006. Only the files in the "My Documents" folder would be encrypted. Victims had to purchase decryption software from certain websites to access their files. It is fascinating to observe how much of the note from the Archiveus Trojan has been directly appropriated by modern ransomware, including the following passage:

"*Do not try to search for a program that encrypted your information—it simply does not exist in your hard disk anymore. System backup will not help you to restore files. Reporting to police about a case will not help you, they do not know the password. Reporting somewhere about our email account will not help you to restore files. Moreover, you and other people will lose contact with us, and consequently, all the encrypted information."*

Many other ransomwares would appear in waves from that point onward, that would try different approaches to encrypt data and make its recovery more difficult, sometimes even managing to make it virtually impossible. As time passes, more and more ransomware variants are appearing and the damage costs they produce keep increasing. According to [1] and [9], in 2015 ransomware cost the world $325 million, in 2021 around $20 billion and it is expected to rise to $265 billion by 2031.

3. Proposed methodology

The analysis of the file holding the encrypted stolen data is the first stage in the suggested approach, the phases of which are shown in **Figure 2**. This may be done by simply opening it in a hexadecimal editor to see if it is a text file or if there were any patterns that would suggest the file is a binary one, a method of encoding, such as Base64, Radix-64, or something else. Once this verification receives a favorable result, one should go on to the data decoding. Repeat this procedure until it is impossible to recognize an output that contains encoded data.

Compression of the data can be used to determine its amount of redundancy, which can be used to determine whether a traditional (or weak) cryptographic procedure is being employed. A good encryption system should provide random-looking results, which indicates that compression should result in a larger file, instead of a smaller one. This is because compression relies on a small number of items appearing more frequently than others, which should not happen in a random stream when considering a sample of appropriate size because each element typically appears around the same number of times. Making a histogram of the file contents and looking for an uneven distribution of the data is another method to verify this.

The cryptanalysis of a traditional algorithm may be done using a variety of methods. One may use frequency analysis, which is based on certain facts, for basic substitution ciphers: in the ciphertext, plaintext symbol frequencies are preserved, and each language has a unique symbol frequency distribution. Given these details, the concept is straightforward by exchanging similar-frequency characters from one alphabet for another. Language statistics, specifically those pertaining to the frequency of digraphs and trigraphs, can also be used to crack transposition ciphers. Another idea could be the usage of the frequency of diagrams and trigrams in the given language. One can use Kasiski's [10] approach, which considers that a repeating sequence of symbols produces the same ciphertext when encrypted with the same key locations, in the case of polyalphabetic mechanisms. This determination of the key length *k*, which is sufficient to limit the original issue to the cryptanalysis of *k* mono-alphabetic ciphers, is made possible by this observation. As an alternative, the index of coincidence [10], which gauges the relative frequency of symbols in the ciphertext, can be used to determine the period of the polyalphabetic cipher.

Typically, the encryption algorithm that produced a specific ciphertext should not be identifiable. However, by looking at the encrypted data, one can at least attempt to gather some information about the type of cipher that was used. One way to do that is by searching for known structures that might be used in different ciphers. For instance, *PC1* and *PC2* matrices are often defined by DES [10] implementations to be used in the key scheduling procedure, or the forward or inverse *S-Boxes* matrices definitions, which are already in place, if we use AES [10] as another example. From here the key can be found by looking at the code referencing that data. Another way to find the key would be looking for it in the binary of the data. Hiding sensitive information in the source code is surely a vulnerability, but malware creators usually do that. If this fails, Shamir’s technique [7] can be used, which considers the entropy of a securely generated key. The idea is to look through a fixed-size window for an area that has the greatest entropy by scanning the entire binary.

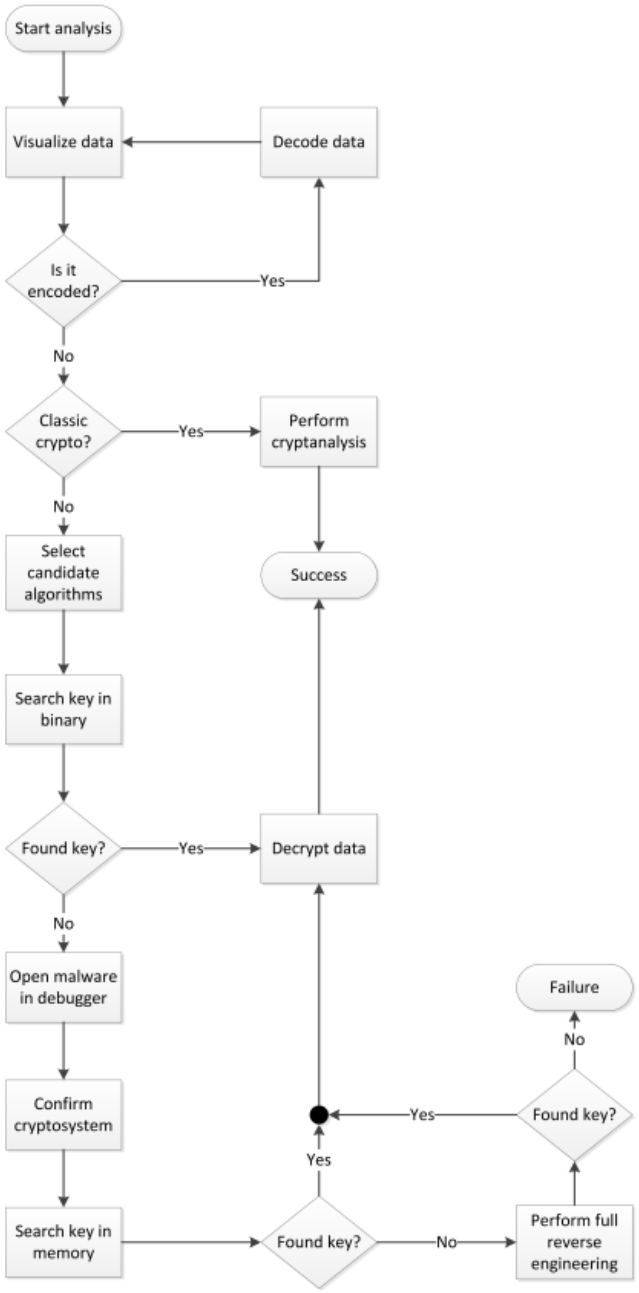

Figure 2. Proposed methodology (see [6])

4. Case studies

4.1. First Malware

The malware presented in this part only uses classical cryptography, therefore, to recover the original data we only need to examine the output file. The name of this malware is **systen.exe** because this is the name of the file found on its victim's computer.

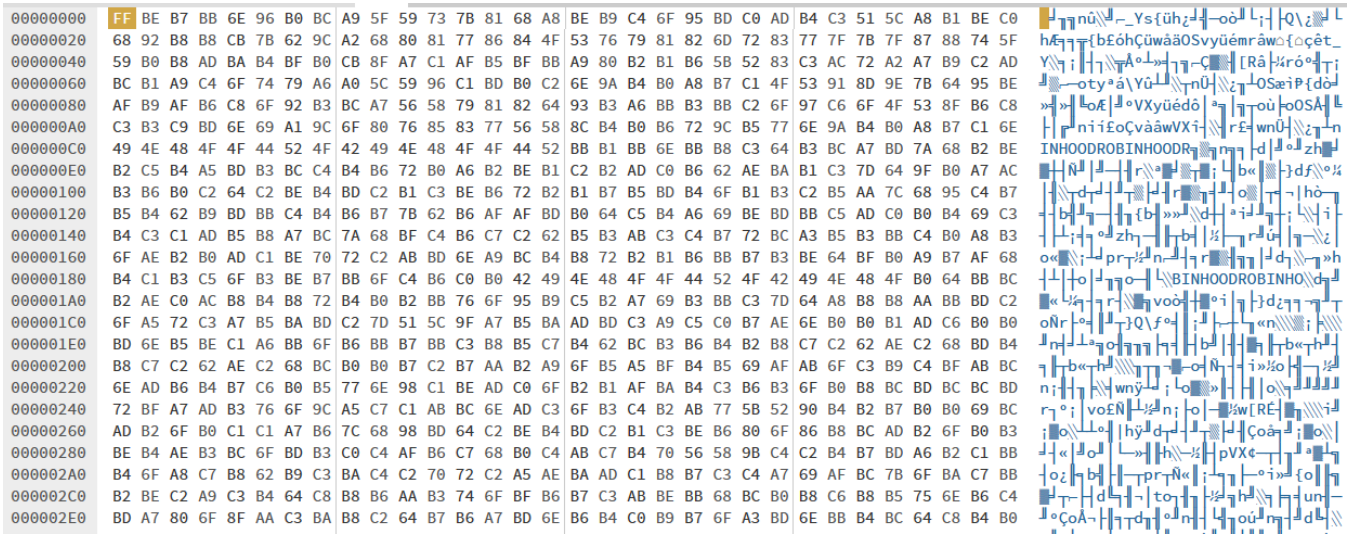

Figure 3. Encrypted sample of "system.exe"

As was already mentioned, one of the fundamental principles of a good encryption algorithm is that its output should appear random, meaning that there should be no patterns in the ciphertext. The encrypted file in this section does not comply with this requirement, which can be seen by the repeated appearances of "ROBINHOOD". The histogram shown in Figure 3 clearly demonstrates a non-uniform distribution, with values concentrated between 100 and 200. The histogram in Figure 4 may also be used to visually identify the problem.

Now that we have established that a classic cryptographic scheme is used, we need to find out which one. The first thing that might come to mind is that a constant number modulo 256 is being added to each byte. We can check this, but it will not produce any meaningful output. By looking at the repeating text "ROBINHOOD", we see that it appears 198 (0xC6) bytes from the beginning of the file and 225 bytes after the first appearance, on position 0x195. Both 198 and 225 are divisible by 9, the length of the text. This might mean that a Vigenère cipher [10] over an alphabet of 256 elements is used as the encryption algorithm. From now on we can use different methods to break this cipher such as the Kasisky method to find the key length, frequency tables or any other method described in [11]. For our case, it is enough to guess the key by looking at the ciphertext. The text "ROBINHOOD" looks like a good candidate for it since it is very possible that it was added to a series of null bytes from the original text. Testing this theory, we obtain the result shown in Figure 5.

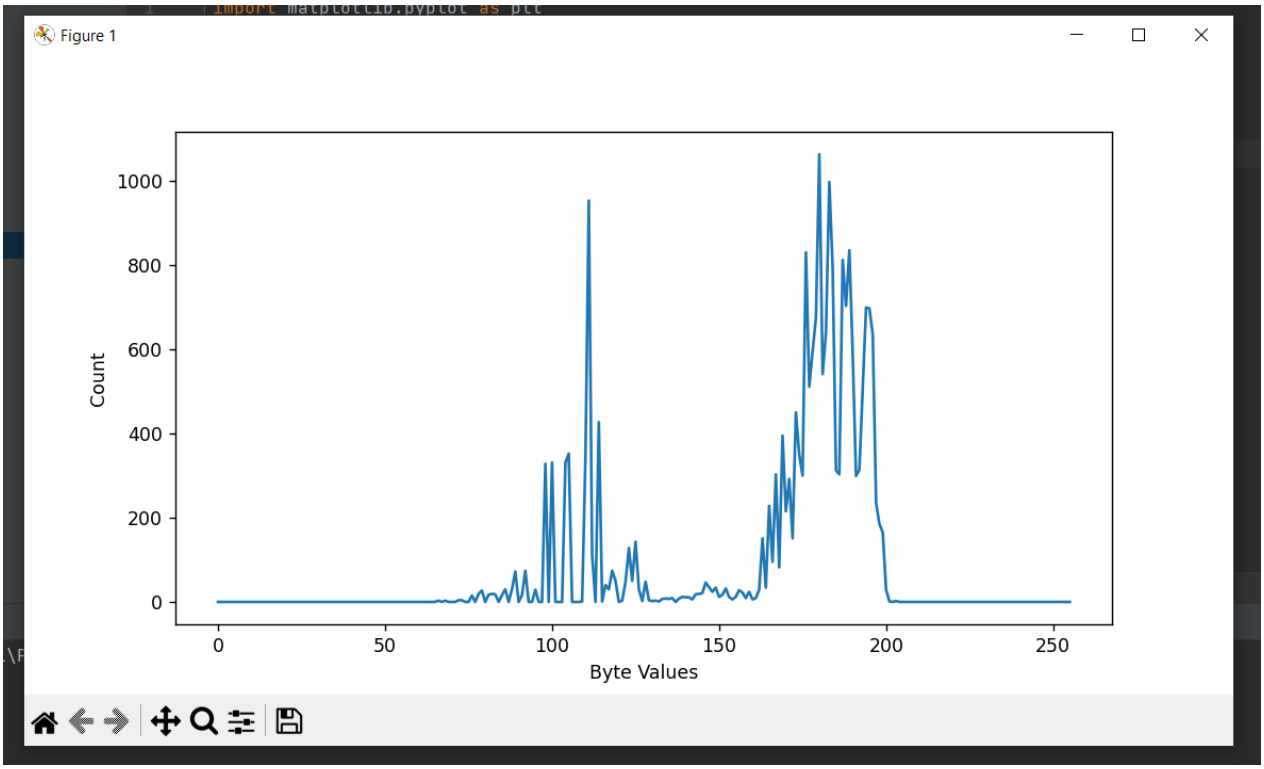

Figure 4. Histogram of byte values for the encrypted file

```
00000000  59 6F 75 72 20 4E 61 6D 65 0D 0A 31 32 33 20 59 6F 75 72 20 53 74 72 65 65 74 0D 0A 59 6F 75 72  Your Name..123 Your Street..Your
00000020  20 43 69 74 79 2C 20 53 54 20 31 32 33 34 35 0D 0A 28 31 32 33 29 20 34 35 36 2D 37 38 39 30 0D   City, ST 12345..(123) 456-7890.
00000040  0A 6E 6F 5F 72 65 70 6C 79 40 65 78 61 6D 70 6C 65 2E 63 6F 6D 0D 0A 34 74 68 20 53 65 70 74 65  .no_reply@example.com..4th Septe
00000060  6D 62 65 72 20 32 30 58 58 0D 0A 52 6F 6E 6E 79 20 52 65 61 64 65 72 0D 0A 43 45 4F 2C 20 43 6F  mber 20XX..Ronny Reader..CEO, Co
00000080  6D 70 61 6E 79 20 4E 61 6D 65 0D 0A 31 32 33 20 41 64 64 72 65 73 73 20 53 74 20 0D 0A 41 6E 79  mpany Name..123 Address St ..Any
000000A0  74 6F 77 6E 2C 20 53 54 20 31 32 33 34 35 0D 0A 44 65 61 72 20 4D 73 2E 20 52 65 61 64 65 72 2C  town, ST 12345..Dear Ms. Reader,
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 6C 6F 72 20 73 69 74 20 61 6D 65 74 2C 20 63 6F  ................lor sit amet, co
000000E0  6E 73 65 63 74 65 74 75 65 72 20 61 64 69 70 69 73 63 69 6E 67 20 65 6C 69 74 2E 20 4D 61 65 63  nsectetuer adipiscing elit. Maec
00000100  65 6E 61 73 20 70 6F 72 74 74 69 74 6F 72 20 63 6F 6E 67 75 65 20 6D 61 73 73 61 2E 20 46 75 73  enas porttitor congue massa. Fus
00000120  63 65 20 70 6F 73 75 65 72 65 2C 20 6D 61 67 6E 61 20 73 65 64 20 70 75 6C 76 69 6E 61 72 20 75  ce posuere, magna sed pulvinar u
00000140  6C 74 72 69 63 69 65 73 2C 20 70 75 72 75 73 20 6C 65 63 74 75 73 20 6D 61 6C 65 73 75 61 64 61  ltricies, purus lectus malesuada
00000160  20 6C 69 62 65 72 6F 2C 20 73 69 74 20 61 6D 65 74 20 63 6F 6D 6D 6F 64 6F 20 6D 61 67 6E 61 20   libero, sit amet commodo magna
00000180  65 72 6F 73 20 71 75 69 73 20 75 72 6E 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 61 20 69 6D  eros quis urna..............a im
000001A0  70 65 72 64 69 65 74 20 65 6E 69 6D 2E 20 46 75 73 63 65 20 65 73 74 2E 20 56 69 76 61 6D 75 73  perdiet enim. Fusce est. Vivamus
000001C0  20 61 20 74 65 6C 6C 75 73 2E 0D 0A 50 65 6C 6C 65 6E 74 65 73 71 75 65 20 68 61 62 69 74 61 6E   a tellus...Pellentesque habitan
000001E0  74 20 6D 6F 72 62 69 20 74 72 69 73 74 69 71 75 65 20 73 65 6E 65 63 74 75 73 20 65 74 20 6E 65  t morbi tristique senectus et ne
00000200  74 75 73 20 65 74 20 6D 61 6C 65 73 75 61 64 61 20 66 61 6D 65 73 20 61 63 20 74 75 72 70 69 73  tus et malesuada fames ac turpis
00000220  20 65 67 65 73 74 61 73 2E 20 50 72 6F 69 6E 20 70 68 61 72 65 74 72 61 20 6E 6F 6E 75 6D 6D 79   egestas. Proin pharetra nonummy
00000240  20 70 65 64 65 2E 20 4D 61 75 72 69 73 20 65 74 20 6F 72 63 69 2E 0D 0A 41 65 6E 65 61 6E 20 6E   pede. Mauris et orci...Aenean n
00000260  65 63 20 6C 6F 72 65 6D 2E 20 49 6E 20 70 6F 72 74 74 69 74 6F 72 2E 20 44 6F 6E 65 63 20 6C 61  ec lorem. In porttitor. Donec la
00000280  6F 72 65 65 74 20 6E 6F 6E 75 6D 6D 79 20 61 75 67 75 65 2E 0D 0A 53 75 73 70 65 6E 64 69 73 73  oreet nonummy augue...Suspendiss
000002A0  65 20 64 75 69 20 70 75 72 75 73 2C 20 73 63 65 6C 65 72 69 73 71 75 65 20 61 74 2C 20 76 75 6C  e dui purus, scelerisque at, vul
000002C0  70 75 74 61 74 65 20 76 69 74 61 65 2C 20 70 72 65 74 69 75 6D 20 6D 61 74 74 69 73 2C 20 6E 75  putate vitae, pretium mattis, nu
000002E0  6E 63 2E 20 4D 61 75 72 69 73 20 65 67 65 74 20 6E 65 71 75 65 20 61 74 20 73 65 6D 20 76 65 6E  nc. Mauris eget neque at sem ven
00000300  65 6E 61 74 69 73 20 65 6C 65 69 66 65 6E 64 2E 20 55 74 20 6E 6F 6E 75 6D 6D 79 2E 0D 0A 46 75  enatis eleifend. Ut nonummy...Fu
00000320  73 63 65 20 61 6C 69 71 75 65 74 20 70 65 64 65 20 6E 6F 6E 20 70 65 64 65 2E 20 53 75 73 70 65  sce aliquet pede non pede. Suspe
00000340  6E 64 69 73 73 65 20 64 61 70 69 62 75 73 20 6C 6F 72 65 6D 20 70 65 6C 6C 65 6E 74 65 73 71 75  ndisse dapibus lorem pellentesqu
00000360  65 20 6D 61 67 6E 61 2E 20 49 6E 74 65 67 65 72 20 6E 75 6C 6C 61 2E 0D 0A 44 6F 6E 65 63 20 62  e magna. Integer nulla...Donec b
00000380  6C 61 6E 64 69 74 20 66 65 75 67 69 61 74 20 6C 69 67 75 6C 61 2E 20 44 6F 6E 65 63 20 68 65 6E  landit feugiat ligula. Donec hen
000003A0  64 72 65 72 69 74 2C 20 66 65 6C 69 73 20 65 74 20 69 6D 70 65 72 64 69 65 74 20 65 75 69 73 6D  drerit, felis et imperdiet euism
000003C0  6F 64 2C 20 70 75 72 75 73 20 69 70 73 75 6D 20 70 72 65 74 69 75 6D 20 6D 65 74 75 73 2C 20 69  od, purus ipsum pretium metus, i
000003E0  6E 20 6C 61 63 69 6E 69 61 20 6E 75 6C 6C 61 20 6E 69 73 6C 20 65 67 65 74 20 73 61 70 69 65 6E  n lacinia nulla nisl eget sapien
00000400  2E 20 44 6F 6E 65 63 20 75 74 20 65 73 74 20 69 6E 20 6C 65 63 74 75 73 20 63 6F 6E 73 65 71 75  . Donec ut est in lectus consequ
00000420  61 74 20 63 6F 6E 73 65 71 75 61 74 2E 0D 0A 45 74 69 61 6D 20 65 67 65 74 20 64 75 69 2E 20 41  at consequat...Etiam eget dui. A
00000440  6C 69 71 75 61 6D 20 65 72 61 74 20 76 6F 6C 75 74 70 61 74 2E 20 53 65 64 20 61 74 20 6C 6F 72  liquam erat volutpat. Sed at lor
00000460  65 6D 20 69 6E 20 6E 75 6E 63 20 70 6F 72 74 61 20 74 72 69 73 74 69 71 75 65 2E 0D 0A 50 72 6F  em in nunc porta tristique...Pro
00000480  69 6E 20 6E 65 63 20 61 75 67 75 65 2E 20 51 75 69 73 71 75 65 20 61 6C 69 71 75 61 6D 20 74 65  in nec augue. Quisque aliquam te
000004A0  6D 70 6F 72 20 6D 61 67 6E 61 2E 20 50 65 6C 6C 65 6E 74 65 73 71 75 65 20 68 61 62 69 74 61 6E  mpor magna. Pellentesque habitan
```

Figure 5. Decrypting the sample with key “ROBINHOOD”

4.2 Second Malware

The second malware is called **TorrentLocker**, also known as **Crypt0L0cker** [13]. It is a ransomware tool that encrypts files and targets all versions of Windows, including Windows XP, Windows Vista, Windows 7, and Windows 8 and it was released towards the end of August 2014. After encryption, a ransom note like the one in Figure 6 will pop up on the victim's machine. Starting at roughly $550 USD, the ransom increases after about three days. Each infected user has a different bitcoin address to which the ransom must be paid.

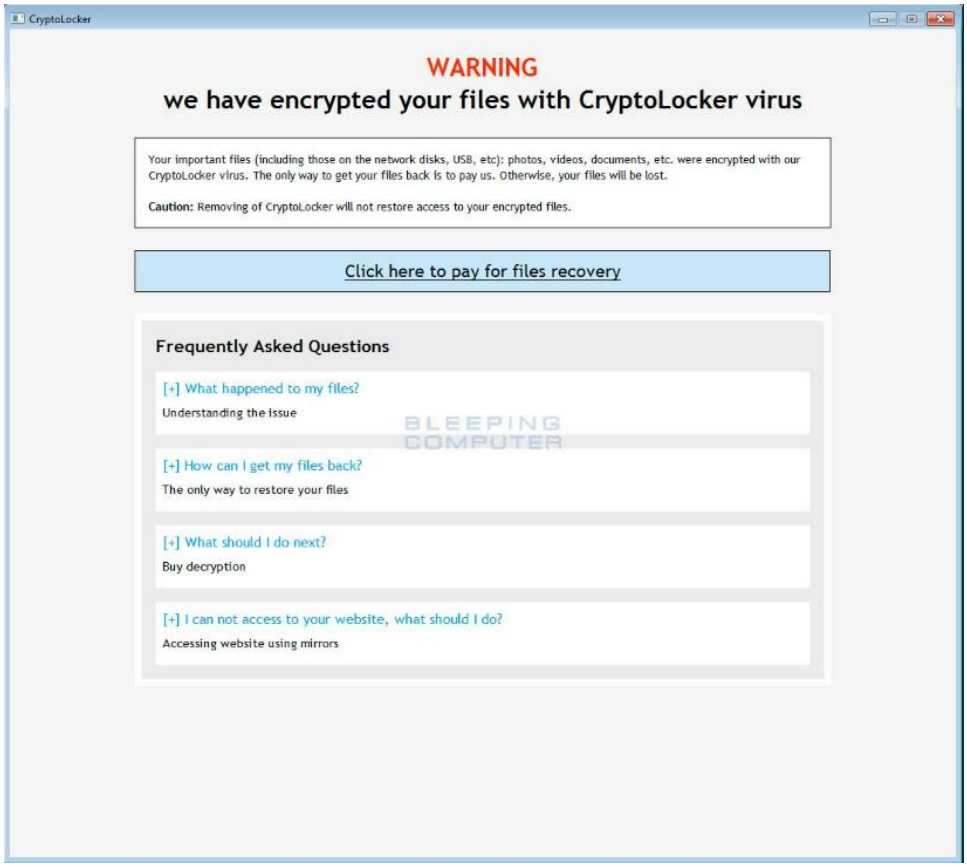

Figure 6. TorrentLocker ransom note

By just looking at the note we are not able to tell how the files are encrypted since it does not tell us anything about it. The encrypted files suggest that a strong encryption algorithm is being used such as AES, DES etc., because of the randomness of the text (Figure 7).

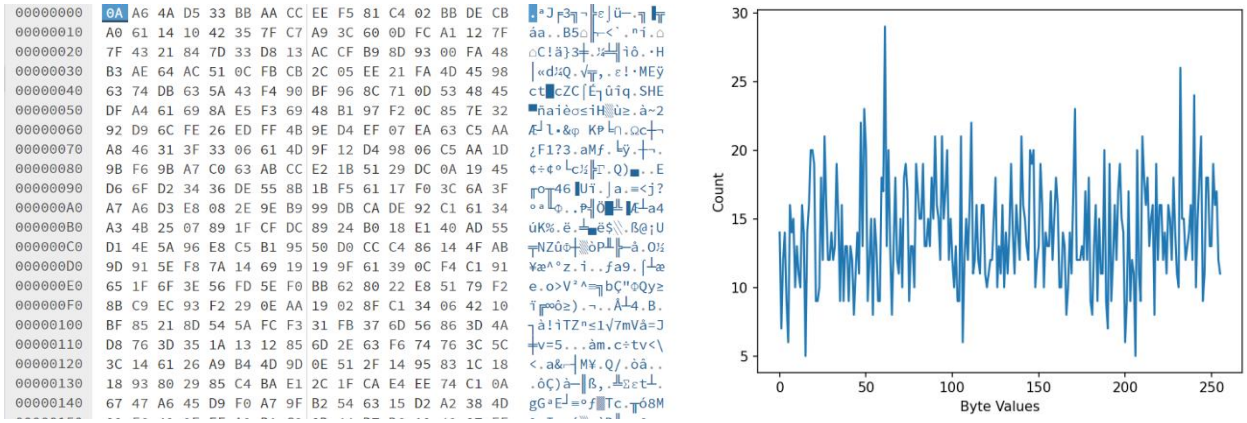

Figure 7. TorrentLocker encrypted file sample and byte distribution

The only option we have now is to look through the binaries of the malware to try to find out how the encryption is done. If we look at the strings found throughout the binary, we can see references to the encryption process. The most valuable information we can extract from this is "tomcrypt\nodes\ctr\ctr_encrypt.c" which is a cryptographic toolkit. The name of the path suggests the use of AES in CTR mode.

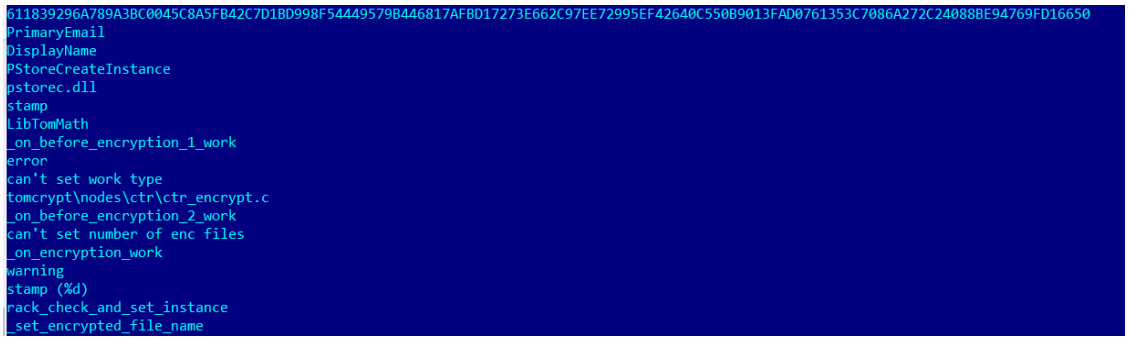

Figure 8. Strings found in the malware binaries

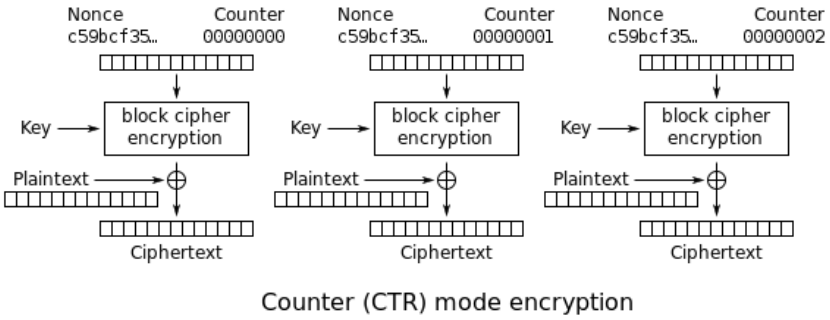

Figure 10. AES CTR mode encryption (see [14])

We can notice another important thing if we look at other encrypted files. As seen in Figure 9, multiple files seem to start with the same sequence of bytes. This might be because the encryption uses the same key and nonce to encrypt all the files and the files shown in Figure 10 might be a certain type of files such as executables, images, or anything else that has a certain format.

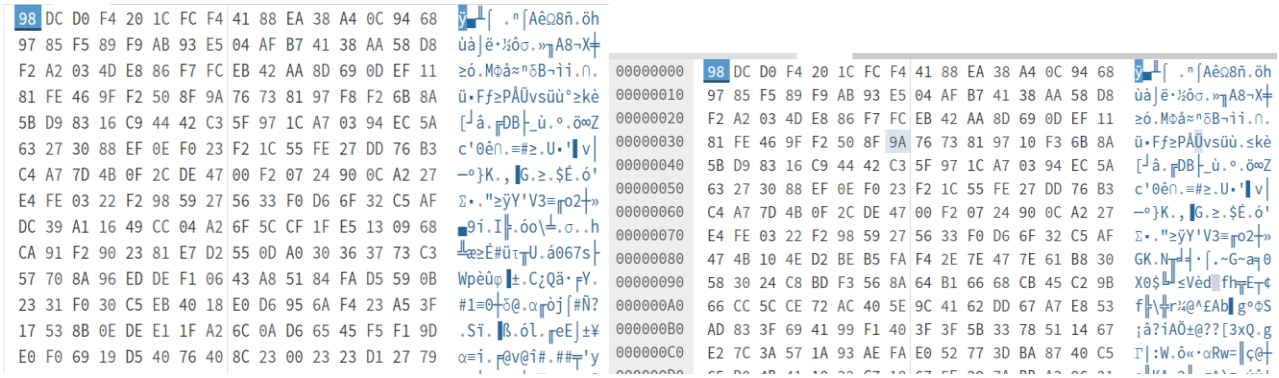

Figure 10. TorrentLocker encrypted samples

If we look at the encryption algorithm described in Figure 9, we can see a vulnerability in the implementation of the ransomware: since the same key and nonce is being used, the algorithm will output the same keystream at every encryption, therefore, we can use the following algorithm to decrypt out files without even knowing the key:

```
A' = ENCRYPT(A)  //the encrypted file
B = large plaintext of NULL bytes (size(B)>size(A))
B' = ENCRYPT(B)

KEYSTREAM = B XOR B'

return A' XOR KEYSTREAM
```

If we follow the described algorithm, we can encrypt out plaintext full of null bytes using the ransomware itself and obtain the following result:

Figure 11. Decrypted file

4.3 Third malware

The third malware is called **portsys.exe** because this is the name of the file this malware is found on the victim's device. In Figure 12 we have a sample encrypted by this ransomware. At first glance it seems that the file is Base64 encoded. If we decode it, we obtain the result from Figure 13 which seems to be encrypted using a strong encryption algorithm.

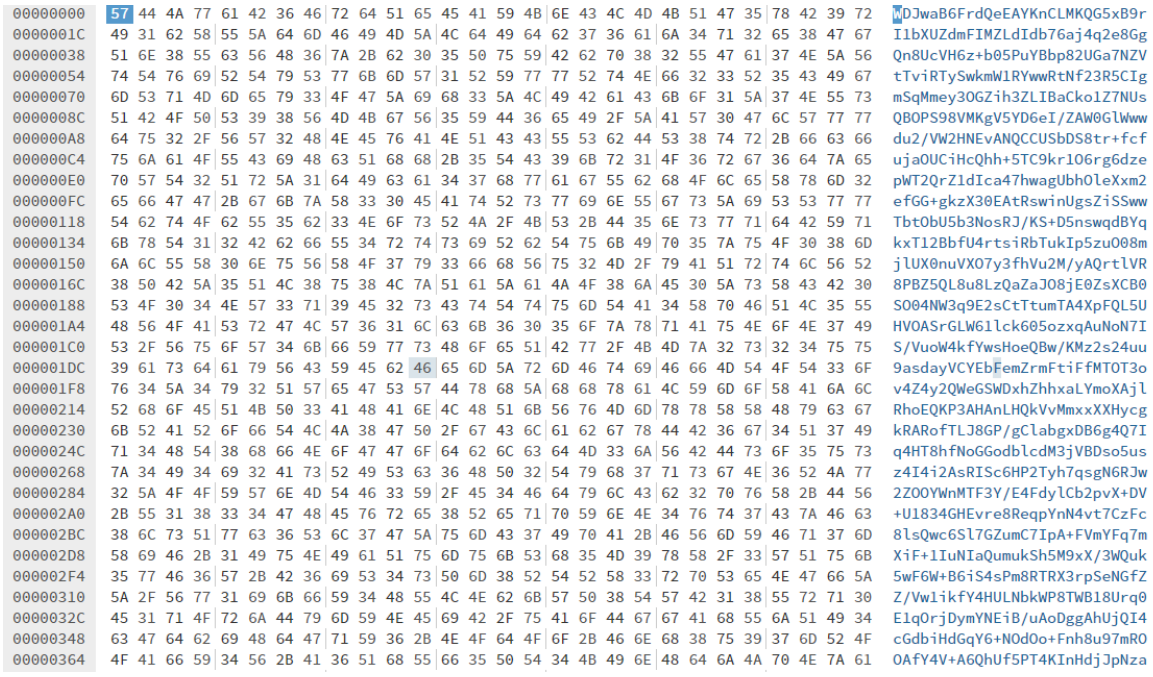

Figure 12. Sample encrypted by portsys.exe

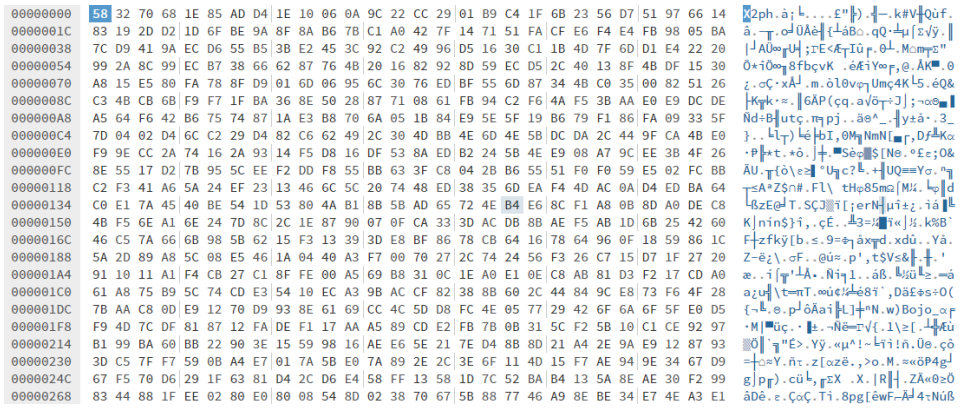

Figure 13. File after Base64 decoding

If we inspect the binaries of the malware, we cannot see anything useful regarding how the file was encrypted. We could try to reverse engineer the file, but this will take some time. What we can do is try and find different constants strong encryption algorithms use such as S-box matrices from AES.

After many searches, we can see the PC1 and PC2 matrices used in DES encryption algorithm as seen in Figure 14.

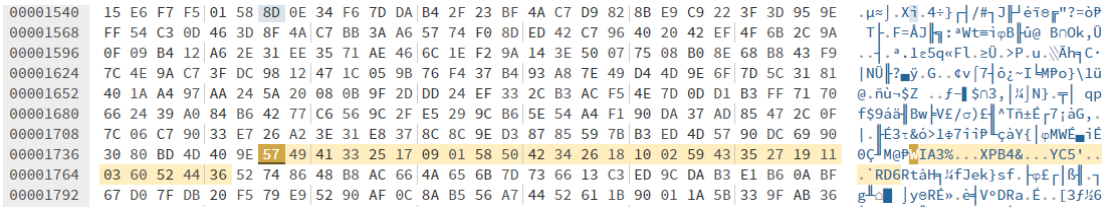

Figure 14. PC1 matrix found in malware binaries

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figure 15. DES matrices extracted from [10]

The next step would be to find the encryption key. Since the PC1 matrix was found hardcoded in the source code of the malware, we might believe that the key is also hardcoded somewhere inside the binaries. As already mentioned, the key should be randomly generated. Starting from this information, an idea that comes to mind is to search for blocks of memory with high entropy. We can identify the locations of the keys by just looking at the data in some suitable way because most of the data in programs have some organization, whereas we anticipate seeing very little structure in key data.

We can find a key by breaking the data into smaller portions, measuring the entropy of each portion, and displaying the areas with very high entropy because we know that key data has more entropy than non-key data. Although obtaining a true measure of entropy is a difficult task, most program code has an entropy level that makes counting the unique bytes in each block a particularly good and simple measurement for entropy. After making some tests, we found that the entropy values for the majority of programs is similar to the one in Figure 16 when taking block of 64 bytes.

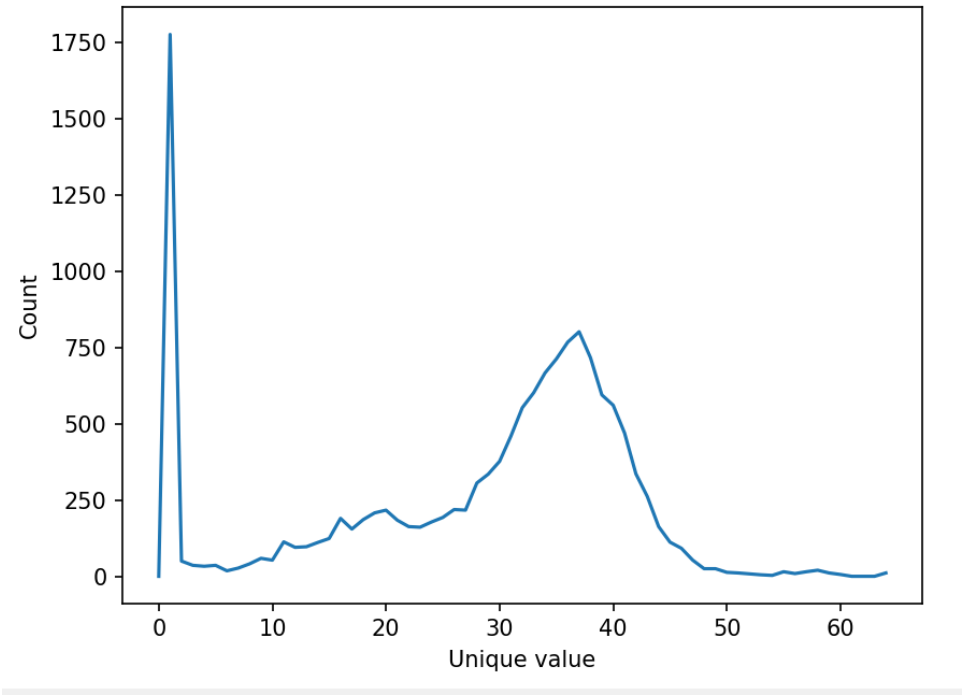

Figure 16. Entropy of the majority of binaries

As we can see, most blocks have around 30-40 unique byte values. By applying this method on our malware sample, we found 23 high entropy blocks, some of them being the blocks that contain PC1 and PC2 matrix, the other containing other forms of random data. By trying to decrypt the sample using different possible keys in these blocks, we found out key that correctly decrypts our file: 0xb34aa010811eb173.

5. Conclusions

In this post, we discussed a method for extracting data from malware-generated encrypted files with the least amount of work necessary. Most of the methods employed to achieve that goal rely on cryptanalysis rather than static and dynamic reverse engineering.

However, it should be noted that if cryptography is employed correctly in situations like the ones described above, it is impossible to succeed without having access to a memory dump of the real-world setting. A malware program that creates session keys to encrypt the stolen data and transfers it all together while being secured by a public key cryptosystem is one example. There isn't much that can be done to recover the original data if the criminal is the only one who has the private key. That would require the task of obtaining the data encryption key via cracking a well-known asymmetric cryptosystem.

References

[1] CV-HG-2019-Official-Annual-Cybercrime-Report.pdf (herjavecgroup.com)

[2] sophos-state-of-ransomware-2021-wp.pdf

[3] Datto2019_StateOfTheChannel_RansomwareReport.pdf

[4] The State of Ransomware in 2021 | BlackFog

[5] The History of Ransomware? Understand | Prevent | Recover

[6] “A Methodology for Retrieving Information from Malware Encrypted Output Files: Brazilian Case Studies”, Future Internet 2013, 5, 140-167; doi:10.3390/fi5020140, Rua Dr. Ricardo Benetton Martins

[7] “Playing hide and seek with stored keys”, Adi Shamir and Nicko van Someren September 22, 1998

[8] Preparation_Instruction (mecs-press.org)

[9] Global Ransomware Damage Costs Predicted To Exceed $265 Billion By 2031 (cybersecurityventures.com)

[10] Menezes, A.; van Oorschot, P.; Vanstone, S. Handbook of Applied Cryptography, 5th ed.; CRC Press: Boca Raton, FL, USA, 2001

[11] Five-ways-to-crack-a-Vigenere-cipher.pdf (cipherchallenge.org)

[12] https://www.virustotal.com/gui/

[13] https://www.welivesecurity.com/wp-content/uploads/2014/12/torrent_locker.pdf

[14] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Counter_(CTR)