Investigating Key Recovery Approaches against WannaCry Encryption

Project-II (CS47006) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Aditya Choudhary

(20CS10005)

Under the supervision of

Professor Debdeep Mukhopadhyay

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

May 6, 2024

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: May 6, 2024 (Aditya Choudhary)

Place: Kharagpur (20CS10005)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Investigating Key Recovery Approaches against WannaCry Encryption**" submitted by **Aditya Choudhary** (Roll No. 20CS10005) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Date: May 6, 2024

Place: Kharagpur

Professor Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Abstract

Name of the student: **Aditya Choudhary** Roll No: **20CS10005**

Degree for which submitted: **Bachelor of Technology**

Department: **Department of Computer Science and Engineering**

Thesis title: **Investigating Key Recovery Approaches against WannaCry Encryption**

Thesis supervisor: **Professor Debdeep Mukhopadhyay**

Month and year of thesis submission: **May 6, 2024**

The menace of ransomwares continues to threaten modern computing systems causing billions of dollars in damage and rewarding millions to the perpetrators. Cryptographic ransomwares or crypto ransomwares are one class of rensomwares that perform unauthorised encryption of victim's data and demand a ransom in exchange of the decryption keys. In this work, we have explored the idea of memory forensics, which is capturing the running memory of a device and then analyzing the captured output for evidence of malicious software, to extract cryptographic secrets from Wannacry ransomware and decrypt encrypted files. This is done so to explore various methods of retrieving lost data from a ransomware attack, and this method can be extended to other ransomwares as well in the future.

Acknowledgements

I would first like to thank my thesis advisor Prof. Debdeep Mukhopadhyay. He was very approachable whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed me to make mistakes, but steered me in the right the direction whenever he thought I needed it. I am very grateful to him for being patient towards me.

I would also like to think my guide Shubhi ma'am, who was my first point of contact and was always available for me whenever needed. She guided me throughout my project and always gave me fruitful advice whenever I found myself stuck at any step.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.
Aditya Choudhary

Contents

List of Figures

Abbreviations

VM	**V**irtual **M**achine
C2	**C**ommand **A**nd **C**ontrol
RDP	**R**emote **D**esktop **P**rotocol
WCry	**W**annacry
Wnry	**W**annacry
DLL	**D**ynamic **L**inked **L**ibrary

Chapter 1

Introduction

The menace of ransomwares continues to threaten modern computing systems causing billions of dollars in damage and rewarding millions to the perpetrators. Cryptographic ransomwares or crypto ransomwares are one class of rensomwares that perform unauthorised encryption of victim's data and demand a ransom in exchange of the decryption keys. Data recovery after ransomware encryption is a very challenging problem because of a number of factors such as strength of standard cryptographic algorithms and unavailability of data backups.

Ransomwares are causing widespread mayhem and chaos by attacking individuals and organisations and are denying access to sensitive and extremely important data. As per SAKELLARIADIS (2022), there has been a very steep rise in ransomware attacks since 2014, compromising many computer systems. With most research efforts focused on the prevention and detection phases, recovery still remains a major issue following a ransomware attack. This work focuses on the aftermath of a ransomware infection, that is, we assume all preventative measures have failed and the infection is operating on the host.

Our solution is to make use of the fact that most crypto-ransomwares use conventional implementations of encryption algorithms to encrypt files. In these implementations, there are no attempts to conceal the secrets that are exposed during the encryption process on the host. This fact allows us to extract cryptographic secrets such as keys from memory and hence facilitate file recovery.

One of the biggest challenge in this methodology is to figure out the most optimum window for key extraction. It is possible to identify optimum window by using **Reverse Engineering** and is also possible to identify how to extend this window. In the case of WannaCry ransomware, this window can be extended by using a bigger file system, making sure that the asymmetric keys stay for a longer duration in the memory.

1.1 Background

Ransomwares have been a major threat to systems security for over a decade. As a result, several solutions have been proposed against them. Some of the most successful solutions with the ultimate objective of protecting user's data are :-

1. **Backup Solutions :-** Backups are proposed as the ultimate solution against all cryptoviral infections.Thus when backups are available, the victim can simply wipe the machine clean, reinstall the host OS, and load the data back on the system.

2. **Static-signature-based solutions :-** Similar to other malware, ransomware can be identified using static signatures created for the binaries that are integrated into virus definition files used by antivirus solutions. This is a tried-and-tested method of detecting known threats, widely incorporated in anti-viruses. However, this method will fail for any new variant of ransomware.

3. **Dynamic-behavior-based solutions :-** Ransomware performs a series of expected tasks on the host and this constitutes a partially unique dynamic signature that reflects ransomware behavior during its execution. However, this method generated a large number of false positives as applications behaving similarly will also be categorized into ransomwares.

4. **Honeyfile-based approaches :-** Honeyfile-based approaches for intrusion detection was first described by Yuill et al. (2004) where alarms were set off when these bait files were accessed or encrypted on a system.

5. **Cryptography-oriented solutions :-** Since encryption activities are central to the operation of cryptographic ransomware, these approaches largely target deficiencies in the implemented cryptosystem within the ransomware. For instance, a ransomware with a statically embedded symmetric key within the binary will reveal the symmetric key during reverse engineering.

1.2 Key Management in Ransomware

Developing a better understanding of key management in ransomware is a necessary prerequisite to finding weaknesses that can be exploited for defensive purposes. Key management plays a central role in generating, deploying, concealing and securely wiping the ransomware keys that is ultimately held for ransom. There are various key management methods which are available in ransomwares. Some of the most fundamental ones are :-

1. Symmetric Encryption

2. Asymmetric Encryption

3. Hybrid Encryption

Wannacry uses **Hybrid Encryption Model**, which uses symmetric key like AES for individual file encryption and uses an asymmetric key like RSA for encrypting these symmetric keys.

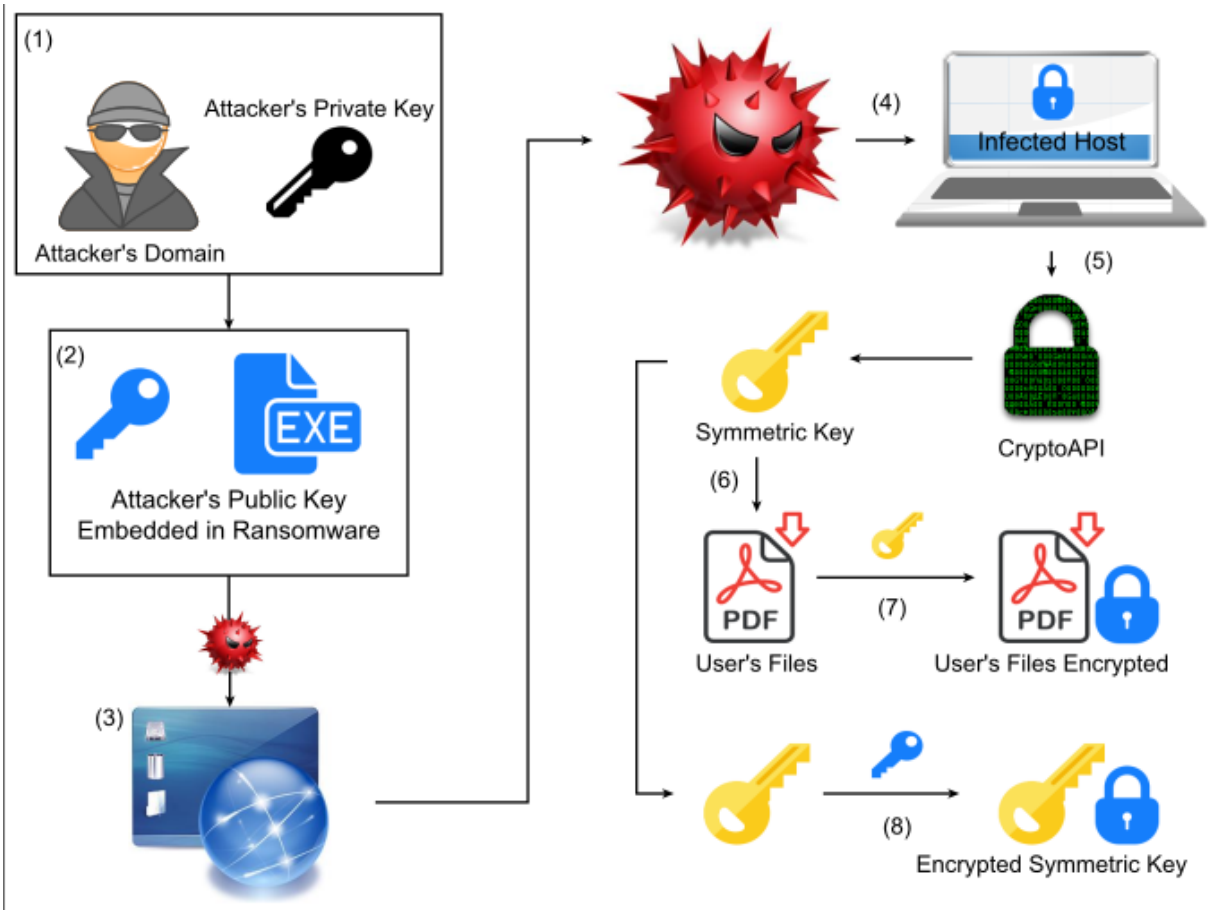

FIGURE 1.1: Basic Hybrid Encryption model in Ransomware, as depicted in Joseph and Norman (2020)

1.3 Conclusion

Our work involves dealing and extracting files after beging encrypted by Wannacry, by making use of memory forensics. Our work involves identifying the most optimum window for key extraction, and identifying ways to extend the window length by using Reverse Engineering WannaCry. We have eventually developed a tool for automated recovery of private key and files from Wannacry.

Chapter 2

Analysing WannaCry

WannaCry is a crypto-ransomware, which targets computers running the Microsoft Windows operating system by encrypting data and demanding ransom payments in the Bitcoin cryptocurrency. It is considered a network worm because it also includes a transport mechanism to automatically spread itself. This transport code scans for vulnerable systems, then uses the **EternalBlue** exploit to gain access, and the **DoublePulsar** tool to install and execute a copy of itself. The systems vulnerable to the ransomware are the ones running Microsoft windows 7 or older, which contains a vulnerable implementation of the Server Message Block (SMB) Protocol. This vulnerability is then exploited by the ransomware worm to spread itself. EternalBlue is the vulnerability in the implementation of SMB protocol, and DoublePulsar is a backdoor tool, which was released by a group of hackers.

My objective in this chapter is to run the ramsomware in a virtual environment and observe its behaviour and changes inflicts in the system. We will also have a close look at its source code to find the information to look for in the dump file of the wannacry process.

2.1 Sandboxing WannaCry

I have used a Windows 7 Ultimate Virtual Machine to sandbox and observe the ransomware. Here, the primary focus is to observe the crypto-ransomware and to note the changes it makes to the victim machine. The worm part of the ransomware makes some network calls to find vulnerable machines in the network. However, observing the worm part is not our objective. Our filesystem contains a photo directory with more than a hundred JPG images, a simple text file and a windows ISO image. The total size of our user filesystem is close to 4 GB.

2.1.1 Attacking Victim VM

Having our file system ready, we can infect the victim machine by running the WannaCry executable on the Desktop. After some time, the victim machine is infected and the following message appears.

FIGURE 2.1: The **WannaDecryptor** interface

Also, the background image of the victim machine is changed to this.

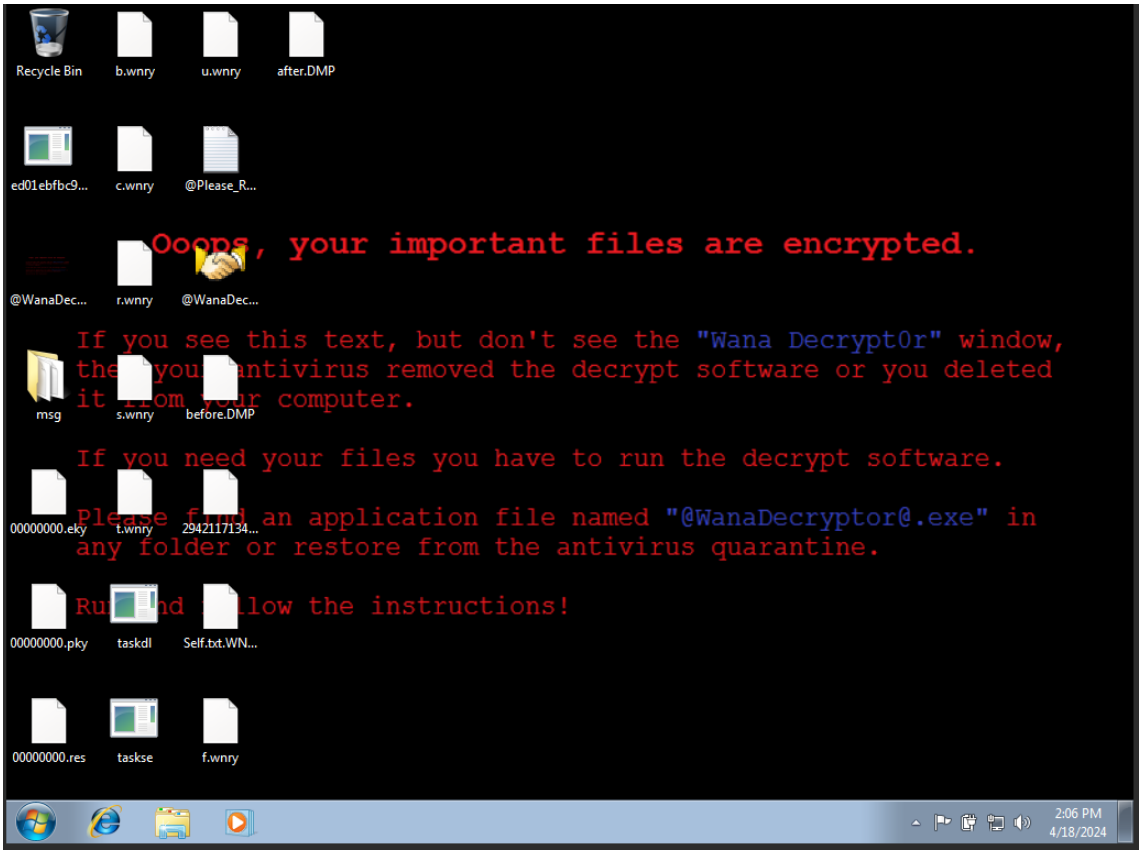

FIGURE 2.2: Change of background image after the machine has been infected

2.1.2 Observations

All of the files in the filesystem have been encrypted, and **.WNCRY** is appended to their file extensions. It can be observed that the WCry process created some files in the same directory in which it is executed. As per Akbanov and Vassilakis (2019), the descriptions of these files have been summarised below.

- **00000000.pky :-** The public victim RSA key. This public key is used to encrypt AES keys used for file encryption. The first 16 bytes of the file contains the **header**, the next 4 bytes consists of the **public exponent** and the remaining data in the file is the **modulus**. The example data of the public key file is as shown below.

FIGURE 2.3: Public Key format. The last 256 bytes (blue portion) of this file is the modulus, and the yellow portion is the public exponent, which is constant and is equal to 65

- **00000000.eky :-** As per Akbanov and Vassilakis (2019), the wannacry executable contains an attacker RSA public key. When the encryption component executes, it generates victim public and private key. The .eky file contains the victim's private key encrypted by the attacker public key. This file is made so that the ransomware can decrypt the files when payment is confirmed.

- **00000000.res** This file contains data and timestamp information for C2 communication. C2 communication is a type of malicious communication between a C2 server and malware on an infected host. An example of this can be the transfer of decryption key upon confirmation of payment in bitcoins.

In Microsoft Windows, a resource is an identifiable, read-only chunk of data embedded in an executable file. WannaCry is distributed as an executable file that contains a password-protected ZIP archive in its resources section, among others. When executed, this archive is unpacked (using the password "WNcry@2ol7") in the current directory and contains the following files :-

- **b.wnry :-** Bitmap image used as desktop wallpaper after the encryption process is done. The image is shown in Figure 2.1.
- **c.wnry :-** Configuration containing Tor C2 addresses, Bitcoin addresses, and other data. As we will see in the next section, the bitcoin addresses are randomly selected by the ransomware and written to this file.
- **r.wnry :-** It contains the ransom demand text, which states that the machine has been infected and ask the user to make payment.
- **s.wnry :-** ZIP archive containing Tor software to be installed on the victim's system.
- **t.wnry :-** Encrypted DLL containing file-encryption functionality. The file encryption software is encrypted and stored in this file. The first 16 bytes of this file is the encrypted AES key, which can be decrypted using the attacker public key, which is declared as a global variable in the executable. After AES key is extracted, the remaining of the file can be decrypted to be executed.
- **u.wnry :-** Main module of the WCry ransomware decryptor **WannaDecryptor**
- **taskdl.exe :-** It is a supporting tool for the deletion of files with the .WNCRY extension
- **taskse.exe :-** Program that displays decryptor window to RDP sessions.
- **msg :-** Directory containing ransom demand messages in multiple languages. It isused by **WannaDecryptor** module to display the messages.

2.2 Reverse Engineering WannaCry

In this section, I have tried to reverse engineer some important components of the encryption component of WCry. As a conclusion, I found out the Windows API calls it made for generating RSA keys and analysed what to look for in the memory dump for successful retrieval of the encrypted data without paying the ransom (without getting the attacker private key). As a result, the encryption component of the ransomware could be understood in a clearer fashion. The decompilation has been done using **Ghidra**, which is an open source decompiler.

2.2.1 Inital Stages

The first significant step that the ransomware takes is accessing a hard-coded random url in its source code. If the access to the url fails, it calls the WinMain() function and execution proceeds. It url access is a success, then the ransomware reutrns without doing any damage. This url would eventually become a kill switch of the ransomware.

If the internet access to the url fails, ransomware proceeds further. After that it proceeds as follows :-

1. It gets the executable path and forms a C string by appending -m flag and security to it. The string would become something like *"executable_path -m security"* and then creates a Microsoft Security Center Service 2.0 using these arguments to start on bootup with all permissions. The service executable is the WCry binary. It is done so that the ransomware can ensure persistence in the victim system.

2. It then extracts resource 1831 from the executable. It creates a new file, named C:/WINDOWS/tasksche.exe and loads the resource 1831 into this executable. After loading the resource, it starts a new process with the executable as this

```
12  i = 14;
13  strange_url = s_http://www.iuqerfsodp9ifjaposdfj_004313d0;
14  strange_url_copy = strange_url_buffer;
15  while (i != 0
16                    /* strncpy(strange_url_copy, strange_url, 14) */) {
17    i = i + -1;
18    *(undefined4 *)strange_url_copy = *(undefined4 *)strange_url;
19    strange_url = strange_url + 4;
20    strange_url_copy = strange_url_copy + 4;
21  }
22  *strange_url_copy = *strange_url;
23  InternetOpenA((LPCSTR)0x0,1,(LPCSTR)0x0,(LPCSTR)0x0,0);
24  hInternet_return = InternetOpenUrlA(hInternet,strange_url_buffer,(LPCSTR)0x0,0,0x84000000,0);
25                    /* if the URL request fails */
26  if (hInternet_return == (HINTERNET)0x0) {
27    InternetCloseHandle(hInternet);
28    InternetCloseHandle(0);
29    FUN_00408390();
30    return 0;
31  }
32  InternetCloseHandle(hInternet);
33  InternetCloseHandle(hInternet_return);
34  return 0;
35 }
```

FIGURE 2.4: Ransomware trying to access the kill switch url

newly created and loaded tasksche.exe. We can extract this resource 1831 using **wrestool** and can analyze it further.

2.2.2 Analysing tasksche.exe

2.2.2.1 Service Creation

In this section, we analyse tasksche.exe, which was loaded and started by the worm. Initially, after the execution of WinMain() function, it generates a random string by taking seed as the computer name. After that it checks for any argument that were passed while executing the resource. If there was an argument and it was /i, then WCry creates a hidden directory in C:/Windows/ProgramData and copies itself into it as tasksche.exe. After that, it creates a new Microsoft service with random string as its name and starts itself without any arguments. Here is the decompiled code segment with renamed function names that describes the same.

```
23  GetModuleFileNameA((HMODULE)0x0,&moduleFIleName,520);
24  generate_random_string(&random_string);
25  piVar1 = (int *)__p___argc();
26  if (*piVar1 == 2) {
27    pcVar6 = &valid_args;
28    piVar1 = (int *)__p___argv();
29    iVar4 = strcmp(*(char **)(*piVar1 + 4),pcVar6);
30    if ((iVar4 == 0) && (iVar4 = create_and_change_cwd_to_hidden_directory(0), iVar4 != 0)) {
31      CopyFileA(&moduleFIleName,s_tasksche.exe_0040f4d8,0);
32      DVar2 = GetFileAttributesA(s_tasksche.exe_0040f4d8);
33      if ((DVar2 != 0xffffffff) && (iVar4 = create_or_start_tasksche_service(), iVar4 != 0)) {
34        return 0;
35      }
36    }
37  }
```

FIGURE 2.5: Code for creating and starting new MS service

2.2.2.2 Resource Extraction

If the executable is involved without the /i argument, then a different flow of instructions is followed. WCry then extracts a password protected resource within it, named 2058. The password is passed as an argument to the function that extracts the resource, hence can be seen from the disassembled code. After extracting the resources, we get some files shown in figure 2.6.

After resource extraction, the program then takes 3 hardcoded bitcoin addresses, randomly chooses one from them and writes them to the c.wncry file which has been extracted just before. The following code segment describes the same.

```
adityach-01@aditya01 ~/Downloads/resources file *
1831.bin:    PE32 executable (GUI) Intel 80386, for MS Windows, 4 sections
2058.XIA:    Zip archive data, at least v2.0 to extract, compression method=d
eflate
b.wnry:      PC bitmap, Windows 3.x format, 800 x 600 x 24, image size 144000
0, resolution 3779 x 3779 px/m, cbSize 1440054, bits offset 54
c.wnry:      data
msg:         directory
r.wnry:      ASCII text, with CRLF line terminators
s.wnry:      Zip archive data, at least v1.0 to extract, compression method=s
tore
taskdl.exe: PE32 executable (GUI) Intel 80386, for MS Windows, 4 sections
taskse.exe: PE32 executable (GUI) Intel 80386, for MS Windows, 4 sections
t.wnry:      data
u.wnry:      PE32 executable (GUI) Intel 80386, for MS Windows, 4 sections
```

FIGURE 2.6: Using wrestool, we can extract 2058.XIA from the tasksche.exe and extract the contents using the password pushed on the stack. The contents of the ZIP file with its type are shown in this figure

```
10  local_10[0] = s_13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb_0040f488;
11  local_10[1] = s_12t9YDPgwueZ9NyMgw519p7AA8isjr6S_0040f464;
12  local_10[2] = s_115p7UMMngoj1pMvkpHijcRdfJNXj6Lr_0040f440;
13  iVar1 = read_or_write_to_c.wncry(local_31c,1);
14  if (iVar1 != 0) {
15    iVar1 = rand();
16    strcpy(local_26a,local_10[iVar1 % 3]);
17    read_or_write_to_c.wncry(local_31c,0);
18  }
19  return;
20 }
21
```

FIGURE 2.7: Randomly choosing one of three address and writing to c.wncry file

2.2.2.3 Decrypting and Extracting from t.wnry

After setting the bitcoin addresses, it does a series of very important steps. First it acquires a crypto context and loads the RSA key from the memory. This key can be referred as the attacker public key.

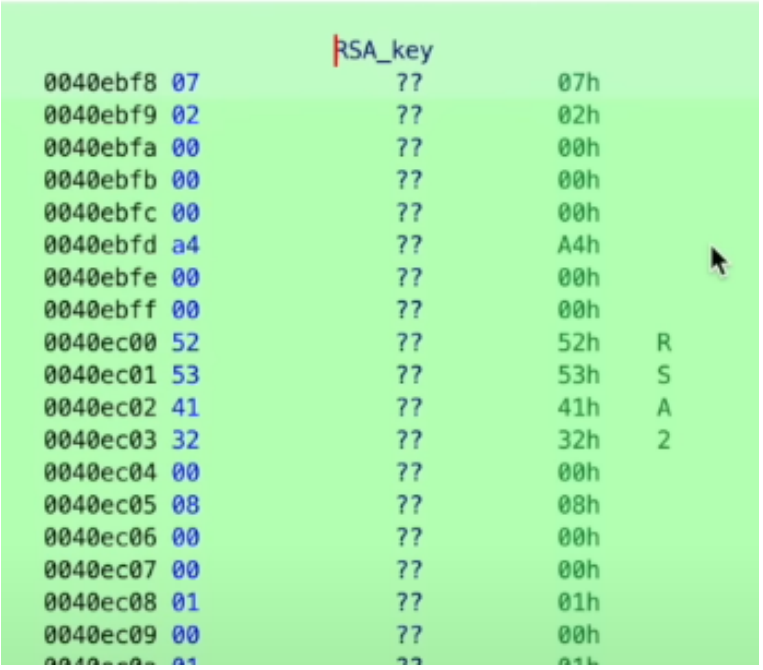

FIGURE 2.8: Attacker Public key as a global buffer in the memory

The purpose of some of the next steps of the program is to extract the DLL from t.wnry and run it by calling a routine from it. It does so by following these steps :-

1. It first loads the first 8 bytes from t.wnry and compares it to the C string **WANACRY!**. If it succeds, it goes to the next steps.

2. It then loads the next 4 bytes and checks if they are equal to 0x100, that is 256. If so, it goes to the next step.

3. It then reads the next 256 bytes from t.wnry into a class. After reading 256 bytes, it ignores the next 4 bytes of the file.

4. Reads another 8 bytes which are given as an argument to GlobalAlloc() function.

5. The 256 bytes which were read into the class are now decrypted using the attacker public key which was loaded before, and the first 16 bytes of the decrypted content are taken as an AES key.

```
{
  LPCRITICAL_SECTION lpCriticalSection;
  BOOL BVar1;

  if (this->rsa_key != 0) {
    lpCriticalSection = (LPCRITICAL_SECTION)&this->mbr_10;
    EnterCriticalSection(lpCriticalSection);
    BVar1 = (*cryptDecrypt)(this->rsa_key,0,1,0,data,&data_size);
    if (BVar1 != 0) {
      LeaveCriticalSection(lpCriticalSection);
      memcpy(data_out,data,data_size);
      *data_out_size = data_size;
      return 1;
    }
    LeaveCriticalSection(lpCriticalSection);
  }
```

FIGURE 2.9: Decryption segment of the taken 256 sized buffer using attacker public RSA Key

6. The 128 bit AES key is then used to decrypt the rest of the t.wnry file (except for the first 280 bytes which were read) to give out a DLL.

7. The control is transferred to the DLL by invoking the entry function TaskStart(). The DLL is the main component of the ransomware responsible for encrypting the file system.

2.2.3 Encryption Process

The encryption component of WannaCry is invoked with the TaskStart system thread, which was extracted from t.wnry as a DLL and invoked by tasksche.exe. During its execution, after performing a few checking steps and changing the current working directory as the directory of tasksche.exe, the encryption process starts.

WannaCry creates three configuration files named **00000000.pky**, **00000000.eky**, **00000000.res**.

Before starting the actual encryption process, it does a bunch of steps, which are as follows :-

1. It checks for the presence of the decryption key file, **00000000.dky** file. If it does not exist, it simply goes to the next step. If such a file exists, it checks for the public key file **00000000.pky**. If both file exists, it imports the keys in both files, which are the encryption and decryption keys respectively. It then encrypts and decrypts test data using these keys and checks if it is same as the original data. If its same, then it spawns a decryption thread, else it goes to the next step.

```
iVar2 = import_key_from_file(this->crypto_provider,&this->key1,pky_file);
if ((iVar2 != 0) &&
   (iVar2 = import_key_from_file(this->crypto_provider,&this->key2,dky_file), iVar2 != 0)) {
  uVar3 = 0xffffffff;
                /* Some strcpy */
  testdata_ = testdata;
```

FIGURE 2.10: Importing the public and private from the files using **CryptImportKey**, which is contained in the implementation of import_key_from_file

```
BVar1 = (*cryptEncrypt)(this->key1,0,1,0,(BYTE *)(testdata + 0xc),&local_22c,0x200);
if ((BVar1 != 0) &&
   (BVar1 = (*cryptDecrypt)(this->key2,0,1,0,(BYTE *)(testdata + 0xc),&local_22c), BVar1 != 0)
   ) {
  uVar3 = 0xffffffff;
  testdata_ = testdata;
```

FIGURE 2.11: Encrypting and Decrypting test data using **CryptEncrypt** and **CryptDecrypt**

2. WannaCry generates a new unique RSA 2048-bit asymmetric key pair, which can be seen in the memory dump made with with SysAnalyzer tool as per Akbanov and Vassilakis (2019).

```
{
  BOOL BVar1;

  BVar1 = (*cryptGenKey)(crypto_provider,1,0x8000001,key);
  return (uint)(BVar1 != 0);
}
```

FIGURE 2.12: The generated RSA key is stored in key argument of **cryptGenKey**

3. It then exports the victim public RSA key to the 00000000.pky file using WinApi's **CryptExportKey** function.

4. It then takes the victim's private RSA key, encrypts it with the attacker's public RSA key in the memory, and then exports it to 00000000.eky file. It is done so that after the payment is verified, the decryption process can be started by decrypting the victim private key using attacker private key.

5. After this, it writes some random data to 00000000.res

6. It then calls destory_keys_and_release_context to delete the keys from the memory and free the memory.

2.2.3.1 File Encryption

After the configuration files have been created and written with appropriate data, the encryption component is ready to start encrypting files on the system. To accomplish this, it spawns 5 threads. Next, one of the threads starts enumerating, every 3 seconds, information about all logical drives attached to the system. If a new attached drive is not a CD ROM drive, then it begins the encryption process on the new drive. At this stage, one of the threads also starts iterating through all existing directories and searching for predefined file extensions of interest.

```
1
2  undefined4 __thiscall OOAnalyzer::cls_1000720c::destroy_keys_and_release_context(cls_1000720c *this)
3
4  {
5    if (this->key1 != 0) {
6      (*cryptDestroyKey)(this->key1);
7      this->key1 = 0;
8    }
9    if (this->key2 != 0) {
10     (*cryptDestroyKey)(this->key2);
11     this->key2 = 0;
12   }
13   if (this->crypto_provider != 0) {
14     CryptReleaseContext(this->crypto_provider,0);
15     this->crypto_provider = 0;
16   }
17   return 1;
18 }
19
```

FIGURE 2.13: Various calls to free context and delete the keys in the memory

```
.der .pfx .key .crt .csr .p12 .pem .odt .ott .sxw .stw .uot .3ds .max .3dm .ods .ots
.sxc .stc .dif .slk .wb2 .odp .otp .sxd .std .uop .odg .otg .sxm .mml .lay .lay6 .asc
.sqlite3 .sqlitedb .sql .accdb .mdb .dbf .odb .frm .myd .myi .ibd .mdf .ldf .sln .suo
.cpp .pas .asm .cmd .bat .ps1 .vbs .dip .dch .sch .brd .jsp .php .asp .java .jar
.class .mp3 .wav .swf .fla .wmv .mpg .vob .mpeg .asf .avi .mov .mp4 .3gp .mkv .3g2
.flv .wma .mid .m3u .m4u .djvu .svg .psd .nef .tiff .tif .cgm .raw .gif .png .bmp .jpg
.jpeg .vcd .iso .backup .zip .rar .tgz .tar .bak .tbk .bz2 .PAQ .ARC .aes .gpg .vmx
.vmdk .vdi .sldm .sldx .sti .sxi .602 .hwp .snt .onetoc2 .dwg .pdf .wk1 .wks .123 .rtf
.csv .txt .vsdx .vsd .edb .eml .msg .ost .pst .potm .potx .ppam .ppsx .ppsm .pps .pot
.pptm .pptx .ppt .xltm .xltx .xlc .xlm .xlt .xlw .xlsb .xlsm .xlsx .xls .dotx .dotm
.dot .docm .docb .docx .doc
```

FIGURE 2.14: File extensions which WCry encrypts

To encrypt each file, it generates a 16-byte symmetric AES key using the **CryptGenRandom** function. Then, it encrypts every generated AES key with the victim's public RSA key and stores it inside the file header starting with the **WANACRY!** string value. Encrypted files are renamed and appended with the .WNCRY file extension.

2.2.3.2 File Decryption

To decrypt the files, we need the victim's private key to decrypt the AES keys used to encrypt the files. After getting the AES keys, the decryption is striaght forward. To get the victim's private key, we have the following ways :-

- Pay the ransom to the attackers and get the attacker private key, which will be used by the WannaDecryptor to get the victim's private key from the 00000000.eky file and generate the decryptor file 00000000.dky.
- Analyze the memory dump of the WCry process and get the victim's private key from there and generate the decryptor file. We have developed a way for achieveing this.

2.2.4 Conclusion

In conclusion we have the following points :-

- The ransomware used Window's CryptoAPI for symmetric and asymmetric key generation. Hence, inspecting the memory for cryptographic secrets can be a successful approach, subject to the fact that we know the correct window for inspection.
- Window's CryptoAPI functions **CryptReleaseContext** and **CryptDestroyKey** do not destroy the key in memory until the associated memory has been freed. Hence, if memory freeing can be prevented, we may have keys in the memory for a larger duration of time.
- After freeing and destroying the keys, the 5 encryption threads are spawned. These threads are responsible for file encryption, and prevents associated memory from being freed. Hence, having a larger file system can ensure that we have a larger window for extracting keys from the memory.

```
OOAnalyzer::cls_1000720c::destroy_keys_and_release_context(pcVar2);
(*pcVar2->vfptr_0->VIRT_FUN_10003a40_0)(pcVar2,1);
pvVar3 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,write_time_thread,(LPVOID)0x0,0,
                      (LPDWORD)0x0);
if (pvVar3 != (HANDLE)0x0) {
  CloseHandle(pvVar3);
}
Sleep(100);
pvVar3 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,check_whether_decrypt_thread,
                      (LPVOID)0x0,0,(LPDWORD)0x0);
if (pvVar3 != (HANDLE)0x0) {
  CloseHandle(pvVar3);
}
Sleep(100);
pvVar3 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,lpStartAddress_10005730,(LPVOID)0x0,0
                      ,(LPDWORD)0x0);
Sleep(100);
pvVar4 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,run_taskdl.exe,(LPVOID)0x0,0,
                      (LPDWORD)0x0);
if (pvVar4 != (HANDLE)0x0) {
  CloseHandle(pvVar4);
}
Sleep(100);
pvVar4 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,
                      (LPTHREAD_START_ROUTINE)&lpStartAddress_10004990,(LPVOID)0x0,0,
                      (LPDWORD)0x0);
if (pvVar4 != (HANDLE)0x0) {
  CloseHandle(pvVar4);
}
Sleep(100);
```

FIGURE 2.15: Five encryption threads spawned after destroying keys. However, the keys are present in the memory until the associated memory is not freed

Chapter 3

Dump Analyzer for key extraction

3.1 Motivation

As seen in the previous chapter, vulnerability in the implementations of CryptReleaseContext and CryptDestroyKey makes it possible to extract the key from the memory even after these functions have been called. The main issue is that the CryptDestroyKey and CryptReleaseContext does not erase the prime numbers from memory before freeing the associated memory. This vulnerability exists in windows 7 or earlier systems. Later implementations do clean up the prime numbers, hence analysing dump file to get the prime numbers will not work on later versions of windows. Since the systems vulnerable to WCry attacks are Windows 7 or earlier, hence this method would work on those systems.

The idea is to basically create the dump of the WCry process and expect those primes to still stay in the memory because of the vulnerability. These prime numbers were created during CryptGenKey function call, as shown in figure 2.10, to generate victim's public private key pair. The motive is to develop a software that can create the dump file and look for those prime numbers in that dump file, and if found,

generate the private key (00000000.dky) from those, which can eventually be used for decrypting and recovering files.

3.2 Implementation

As seen in previous chapter, the public key file (00000000.pky) consists of the public exponent (e) and the modulus (N). The modulus is 256 bytes in size, hence prime numbers will be 128 bytes in size. The idea is to create the dump of WCry process and analyze every 128 byte window in the dump file and check if the number in that 128 byte window divides N or not. If the number divides N, then we have one prime number, say p. Other prime number q can be easily found by dividing N with p.

```
Keylen: 256
N:
59 AA 7A BB 31 3F 49 00 11 78 39 58 7A 0B F8 DE
E7 63 EF 20 21 CE EE 70 47 F9 77 E7 7E D8 87 39
DB E6 FE 06 52 A5 2F B8 41 2B 7F 9C F4 93 B9 17
50 64 C5 6E CE 83 5E B0 0C CD C7 F6 1F 03 63 FD
11 E9 73 67 1F E4 6B 02 F1 71 24 44 4D 76 02 03
76 28 6A 35 C9 79 3D 9E 60 D6 0F A5 0D 63 B9 37
97 4B 20 06 77 E4 40 43 80 2C 58 05 BB B8 CB 86
6E 55 AC 8E C2 9A 31 BF A5 24 E0 D8 79 D1 C1 96
43 9F C8 CA 86 17 6A 0A 4C C7 24 4B D2 16 A6 3A
ED F6 7F 12 E5 57 97 F5 F9 6E 7F FF 99 58 B1 C1
C4 85 09 13 DA 1B 40 D4 22 9E 38 97 CF 6A D0 3C
BD 71 AD B5 25 7B C7 CF D3 95 6A F7 31 77 71 64
22 DC 46 99 57 44 CE 06 F2 35 04 28 CF AC E5 CF
D9 45 6A 24 41 44 12 87 9C AE 06 70 5D B7 3D A1
CD C1 A1 45 22 6E 4F CB FC 37 40 24 E3 C4 06 48
FE 7F FF FC D6 A8 F8 28 63 24 57 46 BC D1 B6 BD
```

FIGURE 3.1: Getting the Modolus from the public key file

The implementation make use of **boost::multiprecision** library for handling large numbers. The library contains already overloaded operators that makes it possible to operate on large numbers like normal built in data types. The software creates dump file by first getting the PID of the WCry process and then passing it onto **procdump** executable as argument, and then spawning the dump process.After

getting the prime numbers from the dump file, the software works on creating the decryptor file.

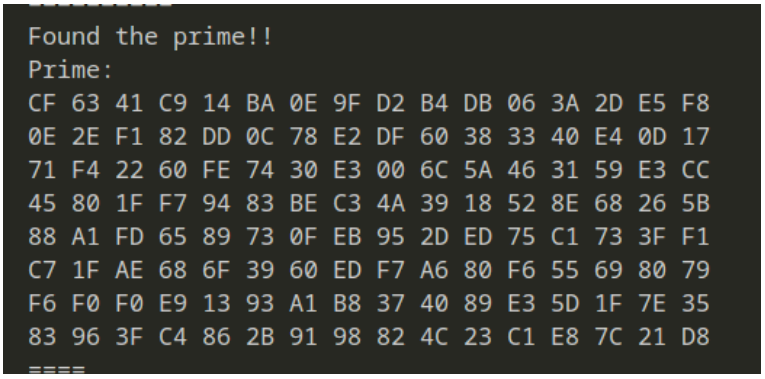

FIGURE 3.2: Finding the prime number from the dump file

3.3 Generating Private Key

The private key file or the decryptor file, **00000000.dky** contains the following segments :-

1. **Heading :-**

   The heading is a 16 byte number denoting encryption method, which is RSA2. This heading can be taken and stored from the public key file and can be written to the decryptor file.

2. **Public Exponent (e) :-**

   The next 4 bytes are the public exponent, which can again be taken from the public key file and written to this file

3. **Modulus (N) :-**

   This is a 256 byte number, available from the public key file.

4. **Prime Number 1 (P):-**

   128 byte number which has been extracted from the dump file.

5. **Prime Number 2 (Q):-**

   128 byte number which has been extracted from the dump file.

6. **Chinese Remainder Theorem Exponent 1 (dP) :-**

   128 byte number which is the remainder between private exponent and P-1

7. **Chinese Remainder Theorem Exponent 2 (dQ) :-**

   128 byte number which is the remainder between private exponent and Q-1

8. **Chinese Remainder Theorem Exponent Coefficient (iQ) :-**

   128 byte number which is the multiplicative inverse of Q and P, or (iQ * Q) mod P = 1

9. **Private Exponent (d) :-**

   256 byte number which is the multiplicative inverse of e and (P-1) * (Q-1)

   Here is the code segment that does the computations and writes the numbers to the private key file. For calculating the multiplicative inverse of a and b, the **mulInv(a,b)** uses Extended Euclidean algorithm.

```
BigIntTy const Q = N / P;
BigIntTy const Phi = boost::multiprecision::lcm(P - 1, Q - 1);
BigIntTy const d = mulInv(e, Phi);
BigIntTy const dP = d % (P - 1);
BigIntTy const dQ = d % (Q - 1);
BigIntTy const iQ = mulInv(Q, P);
writeIntegerToFile(f, N, PrimeSize * 2);
writeIntegerToFile(f, P, PrimeSize);
writeIntegerToFile(f, Q, PrimeSize);
writeIntegerToFile(f, dP, PrimeSize);
writeIntegerToFile(f, dQ, PrimeSize);
writeIntegerToFile(f, iQ, PrimeSize);
writeIntegerToFile(f, d, PrimeSize * 2);
```

FIGURE 3.3: Code for writing various components to the decryptor file

3.4 Decryption

After generating the private key file, which is **00000000.dky**, and placing the file in the same directory as the WCry process, decryption will start after clicking on the **Decrypt Files** button on the WannaDecryptor window. After some time, all the files will be decrypted and their previous extensions will be restored.

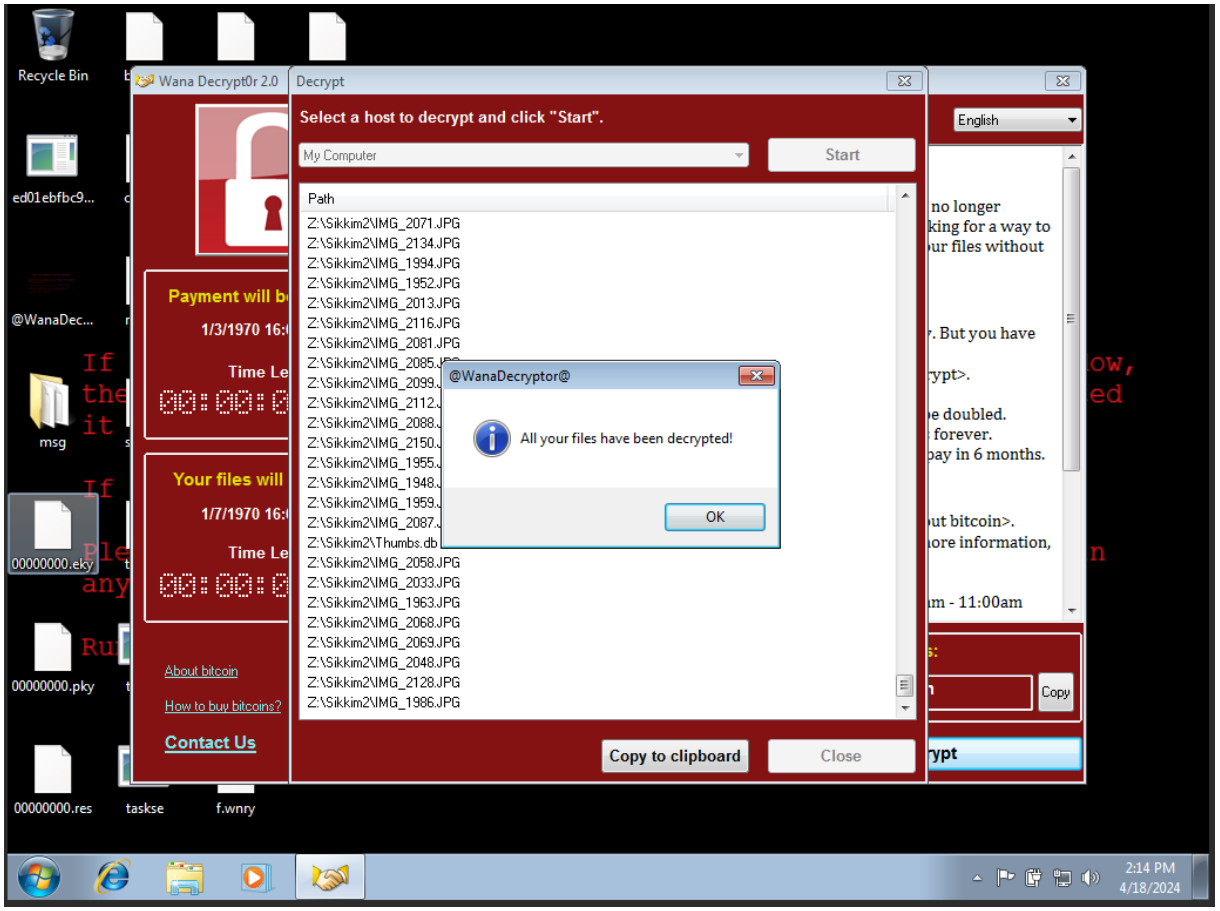

FIGURE 3.4: Successfully decrypting and retrieving files after generating the private key file

Chapter 4

Future Work

In future, few things which can be worked on are :-

- We plan to work on generalising this tool for other ransomware variants as well. Instead of focusing solely on WannaCry, our aim to support a variety of ransomware families such as Locky, CryptoLocker, Ryuk, etc. Each ransomware family uses different encryption algorithms, so our tool would need to be adaptable to extract keys from each type.
- After incorporating various methodologies and making the tool adaptable, it can be pipelined with ransomware detection tools. The tool can analyse the memory dump of any process which the ransomware detector finds malicious.

Bibliography

Akbanov, M. and Vassilakis, V. (2019). Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms. *Journal of Telecommunications and Information Technology*, 1:113–124.

Joseph, D. P. and Norman, J. (2020). *A Review and Analysis of Ransomware Using Memory Forensics and Its Tools*, pages 505–514.

SAKELLARIADIS, J. (2022). Behind the rise of ransomware. Technical report, Atlantic Council.

Yuill, J., Zappe, M., Denning, D., and Feer, F. (2004). Honeyfiles: Deceptive files for intrusion detection. pages 116 – 122.