Assignment3 Team6 Report

Jiayang Xu, Zhengyang Cheng, Haohan Fu, Xibin Yu, Xi Wang and Haoyu Ju

26, March, 2025

1 Analyze the malware's code

1.1 Start

We used Ghidra to analyze the malware code. We found it difficult to find the code that implements the encryption function directly from the entry point, so we started at defined strings in the program. Then we found the AES encryption function **AES_Encrypt_140007080**, whose function call tree is shown in Figure 1. For ease of analysis, we have changed some names in the code and added comments to important sections.

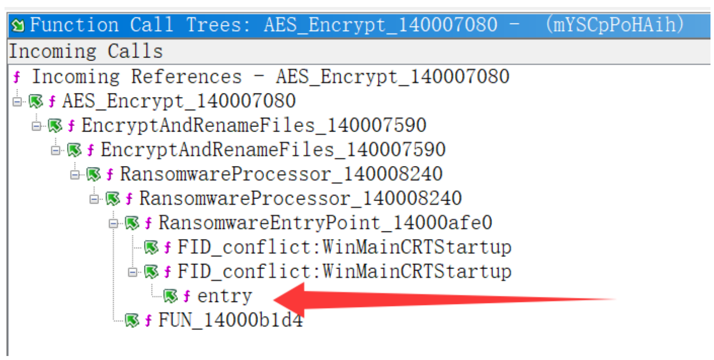

Figure 1: Function call trees

1.2 The AES encryption function in ransomware

The function **AES_Encrypt_140007080** is an AES encryption function, which is the core function of the ransomware, as shown in Appendix A.2.

We analyzed its implementation. The function has two parameters: the path of read the file 'input_path' and the path of write the file 'output_path':

```
void AES_Encrypt_140007080(LPCWSTR input_path,LPCWSTR output_path)
```

It calls some Win32 API functions to read, encrypt and write the target files. The main steps are as follows:

- Open the input file and create the output file by calling **CreateFileW()**.

```
    HANDLE local_f8;
    HANDLE hFile;
    local_f8 = CreateFileW(input_path,1,1,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,(HANDLE)0x0);
    hFile = CreateFileW(output_path,2,1,(LPSECURITY_ATTRIBUTES)0x0,4,0x80(HANDLE)0x0);
```

- Write 16-byte IV to the output file. 'IV_140086010'is a pointer to the memory address of IV.

```
    BOOL BVar2;
    uint lpByteNum [2];
    BVar2 = WriteFile(hFile,IV_140086010,0x10,lpByteNum,(LPOVERLAPPED)0x0)
```

- Read 1008 (0x3f0) bytes sequentially from the input file into the buffer, and write the actual number of bytes read to the output file. In general, the actual number of bytes read is 1008 except for the last one, which may be less than 1008.

```
    undefined8 *buffer;
    HANDLE local_f8;
    uint *local_f0;
    buffer = (undefined8 *)_malloc_base(0x3f0);
    BVar2 = ReadFile(local_f8,buffer,0x3f0,lpByteNum,(LPOVERLAPPED)0x0);
    *local_f0 = lpByteNum[0];
    BVar2 = WriteFile(hFile,local_f0,4,lpByteNum,(LPOVERLAPPED)0x0);
```

- Encrypt the buffer with IV and key by calling **StartEncryption_140008450()**. '0x140086000' is the memory address of the 16-byte key.

```
    undefined keyScheduleWithIV  [192];             /* Address of the key*/
    InitEncryption_140008790((longlong)keyScheduleWithIV,0x140086000,(undefined8 *)IV_140086010);
    StartEncryption_140008450((longlong)keyScheduleWithIV,buffer,0x3f0);
```

- Write encryted buffer to the output file.

```
    BVar2 = WriteFile(hFile,buffer,0x3f0,lpByteNum,(LPOVERLAPPED)0x0);
```

From the above analysis, we were not sure about that the encryption algorithm is AES-CBC-128-NoPadding. Next let's determine the encryption algorithm.

As shown in Listing 1, the first parameter is composed of `key[16] + subkey[160] + IV[16]` (it's no need to analyze the KeyExpansion funtion). So we renamed it to 'keyScheduleWithIV'.

```
void InitEncryption_140008790(longlong keyScheduleWithIV,longlong key_addr,
   undefined8 *IV_address)
{
  KeyExpansion_140009f80(keyScheduleWithIV,key_addr);
  copy_14000c700((undefined8 *)(keyScheduleWithIV + 0xb0),IV_address,0x10);
  return;
}
```

Listing 1: InitEncryption_140008790

In Listing 2 we observed that the **PlaintextXorIV__14000acb0()** function indicates the plaintext xor the IV, proving it's a CBC mode. The **AESOperation__1400088d0()** function implements the AES encryption operation: XorRoundkey, SubBytes, ShiftRows and MixColomns.

```
void StartEncryption_140008450(longlong keyScheduleWithIV,undefined8 *buffer,uint
    size)
{
  /*...*/
  for (i = 0; i < size; i = i + 0x10) {
  PlaintextXorIV_14000acb0((longlong)plaintext,(longlong)IV);
  AESOperation_1400088d0((longlong)plaintext,keyScheduleWithIV);
  IV = plaintext;
  plaintext = plaintext + 2;
}
  /*...*/
}
```

Listing 2: Part of StartEncryption__140008450

To Summarise, we found that the ransomware uses **AES-CBC-128-NoPadding** encryption algorithm to encrypt files.

Note that the decrypted file is not the original file. This is because when encrypting a file, the last encrypted buffer has residual data that has not been overwritten. Therefore, it's necessary to remove the residual data of decrypted data when decrypting.

2 Determine what files/directories are targeted

2.1 Determine what files are targeted

The function **EncryptAndRenameFiles__140007590**, which calls **AES__Encrypt__140007080**, allows us to identify the ransomware target files. This function needs to be passed a path pointer parameter.

```
void EncryptAndRenameFiles_140007590(short *dir)
```

Then the function defines a loop. In this loop, all the files under the 'dir' path will be found by calling **FindFirstFileW()**. 264 (0x104) is the maximum length of the path string: `MAX_PATH = 0x104`.

```
BOOL BVar3;
  do{ /*...*/
      WCHAR local_e88[264];
      ConcatWPath_140007b20(dir, 0x104, (short *)"\\");
      CopyWPath_140007bc0(local_e88, 0x104, dir);
      ConcatWPath_140007b20(local_e88, 0x104, (short *)"*");
      local_1110 = FindFirstFileW(local_e88, &local_10e8);
      /*...*/
      BVar3 = FindNextFileW(local_1110, &local_10e8);
    } while (BVar3 != 0);
```

The loop executes the following code:

- Exclude directories by determining the file attributes (0x10 for directory).

```
    if ((local_10e8.dwFileAttributes & 0x10) == 0){/*...*/}
```

- Compare the first three characters (6 bytes) of the filename with the string ’ en’, and then exclude the file if they are the same.

```
    wchar_t local_e98[8];
    copy_14000c700((undefined8 *)local_e98, (undefined8 *)local_10e8.
    cFileName, 6);
    iVar2 = wcscmp(local_e98, L"~en");
    if (iVar2 != 0){/*...*/}
```

- Retrieve the path of the executable file of the current process (i.e. ransomware) by calling **GetModuleFileNameW()**, and then exclude this file.

```
    WCHAR local_858[264];
    GetModuleFileNameW((HMODULE)0x0, local_858, 0x104);
    _Str2 = PathFindFileNameW(local_858);
    iVar2 = wcscmp(local_10e8.cFileName, _Str2);
    if (iVar2 != 0){/*...*/}
```

- Encrypt the original file by calling **AES_Encrypt_140007080()**.
  Prefix the filename of encrypted file with ’ en’ (e.g. ’sample.md’ to ’ ensample.md’).
  Delete the original file by calling **DeleteFileW()**.

```
    CopyWPath_140007bc0(local_648, 0x104, dir);
    output_addr = local_648;
    ConcatWPath_140007b20(output_addr,0x104,L"~en");
    ConcatWPath_140007b20(output_addr, 0x104, local_10e8.cFileName);
    AES_Encrypt_140007080(input_addr, output_addr);
    DeleteFileW(input_addr);
```

After the above loop is completed, the second loop renames all the encrypted file to the original filename by calling the **MoveFileW()** function (e.g. ’ ensample.md’ to ’sample.md’).

```
    MoveFileW(local_10f0, local_10f8);
```

In summary, for a given directory ’dir’, the ransomware does **NOT** target: subdirectories and files under them, encrypted files and the ransomware itself. However, there is one exception. If a file’s original filename is prefixed with ’ en’, it will not be encrypted, but the file will lose its prefix ’ en’ after the ransomware runs.

But what is ‘the given directory‘? Let’s next identify it.

2.2 Determine what directory is targeted

The function **RansomwareProcessor_140008240** get the current directory by calling the **GetCurrentDirectoryW()** function, and passes it as a parameter to **EncryptAndRenameFiles_140007590()**. Thus, the ransomware only targets the current directory.

```
void RansomwareProcessor_140008240(void)
{
    /*...*/
```

```
    WCHAR dir[264];
    printf((char *)L"Getting current directory. ");
    GetCurrentDirectoryW(0x104, dir);
    EncryptAndRenameFiles_140007590(dir);
    Sleep(10000);
    /*...*/
}
```

Listing 3: Part of RansomwareProcessor_140008240

In conclusion, the ransomware targets: **all the files (not the subdirectories and files under them) in the current directory except the ransomware itself and all files prefixed with ' en'.**

3 Recover the AES key

As noted above, the memory address of the AES key is the second parameter of the function **InitEncryption_140008790**:

```
void InitEncryption_140008790((longlong)context_array,0x140086000,(undefined8 *)
    IV_140086010);                              /*Address of the AES key*/
```

Then we took a screenshot of the key in Ghidra, as shown in Figure 2.

The AES key is '**8d02e65e508308dd743f0dd4d31e484d**'.

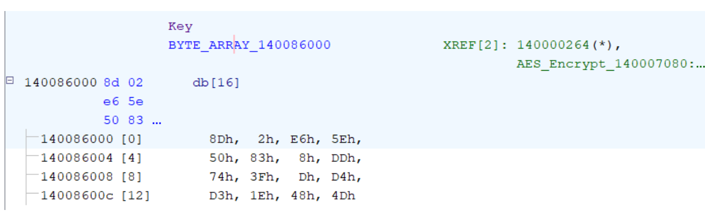

Figure 2: the AES key in Ghidra

4 Decrypt Hank's files.

The tool to decrypt Hank's files is **'assinment3-team6-data/AES_decrypt.py'**.

To use this python tool, first install pycryptodome.

```
pip3 install pycryptodome
```

Then replace the following line with YOUR directory of the files to be decrypted, and DO NOT add a '/' to the end of your directory.

```
FILE_DIRECTORY = "HanksBackup"
```

A Appendix

A.1 The Ghidra zip file

The Ghidra project file is **'assinment3-team6-data/mYSCpPoHAih.gzf'**.

A.2 The C-style decompiled codes

All the complete codes for the above C-style decompilations can be found in the directory **'assinment3-team6-data/C-style decompiled code'**.

In addition, there are some functions not mentioned above, but which are also valuable (because they are part of the function call tree), listed below:

- **entry.c**: it calls the function **RansomwareEntryPoint__14000afe0()**.

```c
void entry(void)
{
    __security_init_cookie();
    RansomwareEntryPoint_14000afe0();
    return;
}
```

Listing 4: entry.c

- **RansomwareEntryPoint__14000afe0.c**: it calls the function **RansomwareProcessor()** mentioned above.

```c
ulonglong RansomwareEntryPoint_14000afe0(void)
{
    /*...*/
            __scrt_get_show_window_mode();
            _get_wide_winmain_command_line();
            /*Ransomware Processor here*/
            uVar3 = RansomwareProcessor();
            uVar7 = __scrt_is_managed_app();
    /*...*/
}
```

Listing 5: Part of RansomwareEntryPoint__14000afe0.c

A.3 The decrypted files

All decrypted files can be found at: `https://github.com/Superior-Josh/FMPT-Assignment3/tree/main/HanksBackup_decrypted`

Academic Conduct & Plagiarism:

We take plagiarism seriously. By submitting your solution, you agree that:

1. The submission is your group's own work and that you have not worked with others in preparing this assignment.

2. Your submitted solutions and report were written by you and **in your own words**, except for any materials from published or other sources which are clearly indicated and acknowledged as such by appropriate referencing.

3. The work is not copied from any other person's work (published or unpublished), web site, book or other source, and has not previously been submitted for assessment either at the University of Birmingham or elsewhere.

4. You have not asked, or paid, others to prepare any part of this work.