Malware analysis

This report covers a detailed analysis of the malware WannaCry.

Casey Donaldson

CMP320: Advance Ethical Hacking

2023/24

Note that Information contained in this document is for educational purposes.

.

Abstract

The increase of malware has posed an increasing threat to several companies and individuals. Malware originated from the Creeper Virus in 1971. The evolution of malware has revealed several different variants including trojans, ransomware, viruses, worms and more. These types of malwares are often deployed with the intent for personal gain, state sponsors and possibly other reasons, with payouts exceeding $40 million. Despite the increase of cyber security, additional tactics are needed to protect against malware. This involves malware analysis; this type of analysis involves analysing the software in a secure environment to identify the functionality of the malware and correct documentation. This helps to create new defences to protect companies and individuals.

During this examination, the malware selected will be unknow until the analysis takes place. This investigation uses a methodology created from two well-known methodologies. The methodology encompasses static and dynamic analysis to examine the malware. Each of these sections contain further sub-sections, like static code analysis where the analyst will examine the application in a debugger to reverse engineer the application, another technique will be packing detection to identify if any packing was used. Each procedure will present results in a relaxed format which should be easy to follow. Furthermore, the results will be spoken and discussed about during the analysis.

Moreover, the aims of this investigation should be met by performing a comprehensive analysis and identify the malware's functionality and document the findings. Finally, the report will bring together the findings to create a conclusion on what was found. Additionally, the malware will be categorized based on the results and countermeasures for the malware will be discusses to avoid the malware in question infecting any further system. Lastly the report will cover future work which will look at the possibility of reverse engineering the decryption key and other types of work which could be conducted.

.

Contents

.

1 INTRODUCTION

1.1 BACKGROUND

Recent years have seen a rapid development and deployment of malware. This has been an increasing concern for an abundance of companies and individuals. Before continuing, readers should understand how malware came to be.

In 1971, a different type of software was introduced. The software created was called the "Creeper Virus". The program did not cause any damage but rather dared the users to catch the creeper and the software quickly infected the systems. This type of software foretold the future of malicious software and revealed the start of malicious code (Radware, 2024). Certain names came around for types of malicious code, a virus was malicious code that propagated through a system with user's interaction. Another example is a trojan which is malicious code designed to look legitimate (Arctic Wolf, 2021). Many other types of malicious code came to be and thus the name Malware came out. This is a term used to describe any type of malicious code (McAfee, 2023). The main purpose of malware is to exploit computer systems and is used typically by cyber criminals.

Malware can be used on a computer system for:

- Data theft
- Corporate espionage
- Cyberwarfare
- Sabotage
- Spyware (spying)
- Ransom for money (Ransomware)
- Control the user's computer
    - DDoS attacks
    - Possibly to mine for cryptocurrency

These can vary and newer malware could be created to perform other malicious activities (AVG, 2015). Depending on the activity, malware used, and the target the rewards can be quite high. The most obvious and common reason why hackers use malware is for personal gain. Either by forcing the company to send money or by selling data which could be valuable. Below is a table, showing the amount of money which can be paid out due to a ransomware attack, See Table 1 Ransomware payouts adapted from (Fortinet, 2024). The amounts within this graph are not the highest payouts but are the most popular attacks known. Some of the payouts from ransomware can reach higher than $40 million.

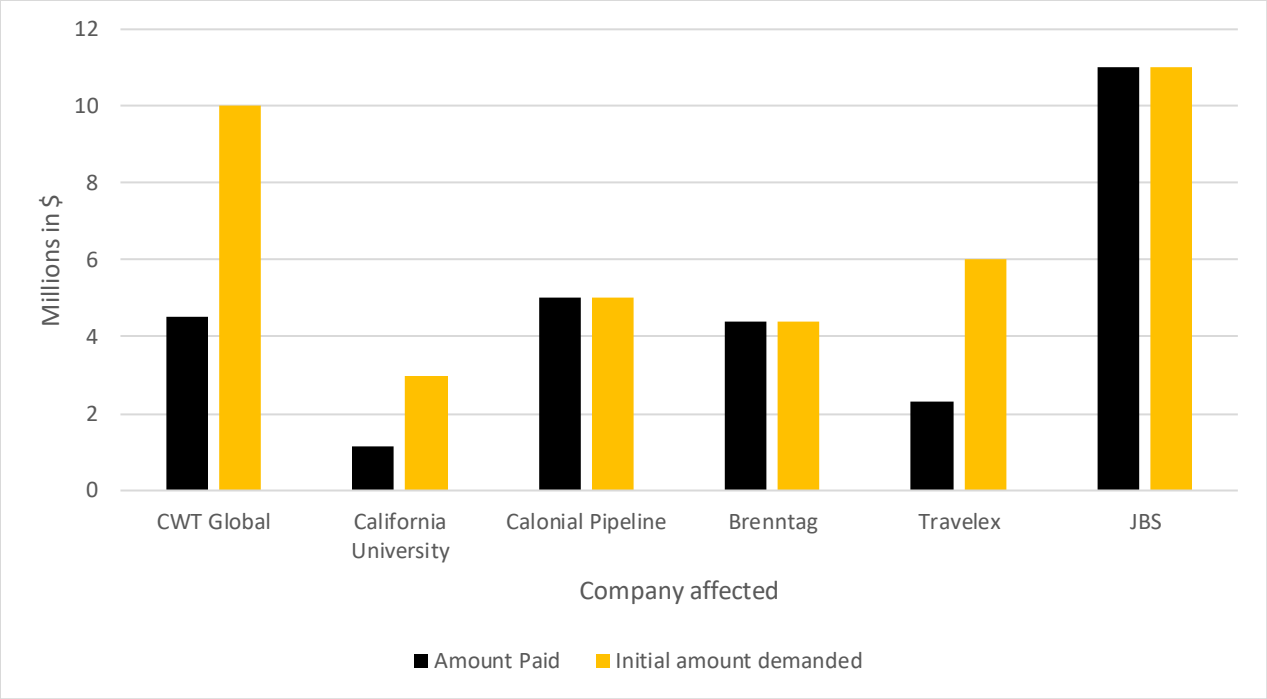

Table 1 Ransomware payouts adapted from (Fortinet, 2024).

However, the malware used differs and uses different techniques to gain access and each malware has its own goal, whether its stealing data, corrupting data, or demanding money. Additionally, every year the total amount of malware has been on an increase since 2008 (AV-TEST, 2019).

In 2010 an interesting malware was discovered called Stuxnet Worm, differing from the report this malware was developed by the US and Israeli government as a state sponsor against Iran. The goal of this malware was to target supervisory control and data acquisition systems, particularly ones related to Iran's nuclear program. With the intentions of being undetected for as long as possible. Furthermore, the malware was extremely sophisticated and used zero-day exploits. Additionally, the malware allowed itself to infect other systems by updating and propagating via USB drives and network shares. The impact Stuxnet had was extremely damaging to Iran's nuclear infrastructure, but it did highlight the increasing use of cyber warfare and increased overall security awareness (Singer, 2015).

Malware would be easy to defend against if it was all the same but new malware is constantly being created and deployed see Table 2 New malware identified, taken from (AV-TEST, 2019). This is why malware analysts exist; they are in charge of finding and analysing the malware.

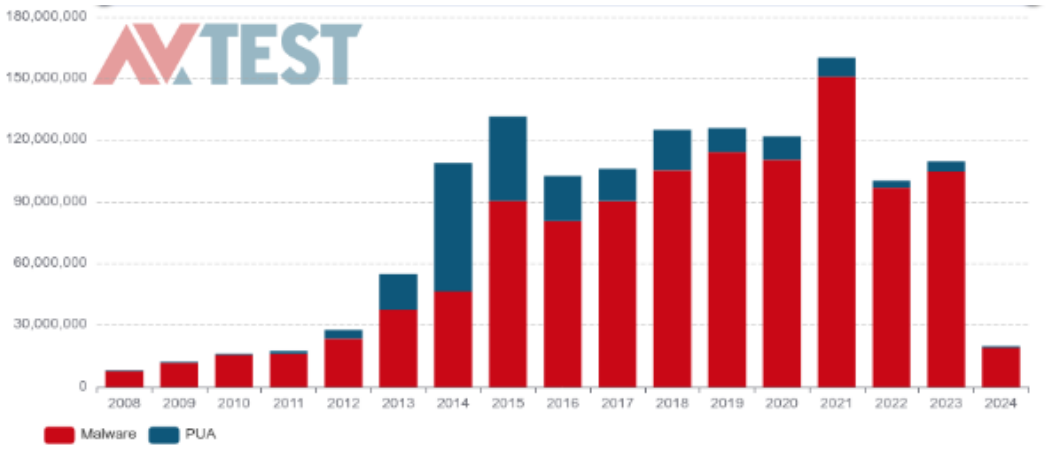

Table 2 New malware identified, taken from (AV-TEST, 2019).

Malware analysis is when an individual will interpret what software is doing, usually malicious software to understand certain aspects of it. This includes the behaviour, potential impact on the system and how the software functions. The process often involves observing the malware, identifying patterns, and examining the assembly code and other techniques. This plays a huge role in defending computer systems. The analysis is often performed by cybersecurity experts, but it can be attempted by anyone.

Malware analysis helps to understand how malware works, this allows people to create and deploy better defences and countermeasures. Malware analysis follows two main approaches, static and dynamic analysis. Static analysis is when the malware is analysed without executing the code. Whereas dynamic analysis, is when the malware must be executed before analysing the results. Both of these methods have different strengths and weaknesses. These two approaches can be further broken down into more in depth techniques; these will be discussed throughout the report.

1.2 AIM

The aims are listed and shown below:

- Perform an analysis on the provided malware.
    - Set up a secure environment to perform analysis on malicious software.
        - Use a virtual machine which is not connected to the network.
    - Static analysis.
        - Perform basic static analysis.
        - Perform static code analysis.
    - Dynamic analysis.
        - Perform behaviour analysis.
        - Perform Memory analysis.
- Report and conclude what the malware does.
    - Document all findings including any discovered indicators of compromise (IOC's).
    - A thorough evaluation of the malware.
    - Appropriate countermeasures to the malware.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

The reason for this analysis is to comprehensively identify and explain the operational mechanisms of the malware. The procedures will follow a methodology designed for this analysis. At a macroscopic level, this methodology will include static and dynamic analysis techniques. Furthermore, the procedures will discuss what was found and present findings throughout this examination. Encapsulated within the overarching discussion will be the conclusion which will be discuss in detail.

2.1.1 Methodology

The methodology used throughout this report was created specifically for this analysis. Researching both SAMA and First.org's malware analysis methodologies revealed similar but different approaches. The below methodology was a combined adaptation of their methodologies, (First.org, n.d.) (Bermejo Higuera et al., 2020).

- Static analysis
    - Basic static analysis
        - This covers basic techniques to find information like imports and compile time.
    - Static code analysis
        - This covers reverse engineering the application to find out how the application runs.
- Dynamic analysis
    - Basic dynamic analysis
        - Observing the malware identifying anything which can be seen by a standard user.
        - Process monitoring to identify all processes which the malware executes and uses.
        - Registry editing involves identifying what registry entries the malware enters, delete, or manipulates.
        - Network analysis involves identify any network traffic generated by the malware.
    - Memory analysis involves analysing the memory statically on an image file where the malware was executed on.

2.1.2 Tools used.

This section will list all the tools used throughout the analysis to allow the readers to replicate the results. Some tools are not mentioned in the report but were used to replicate a result, additionally these tools can be used as an alternative.

- Regshot
- VMware Workstation 17 Pro
- Wireshark
- ProcMon

- Process explorer
- Fakenet
- ApdateDNS
- Windows built-in utilities
- Volatility3
- PE explorer
- Hashmyfiles
- File Hash - MD5 hashing
- VirusTotal.com
- Detect It Easy (DIE)
- Strings.exe
- Ghidra & IDA Pro and the Free version.
- 7 Zip
- TrID
- PEiD
- Upx
- PEview
- Pestudio
- Dependency walker

2.1.3 Background on software given.

The analyst was given a choice of 9 different samples. Sample 1 was chosen for the project. The software given was in the form of an executable/application and the name of the file is: ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe.

2.1.4 Creating and using a secure environment for analysis.

This section explains how the analyst went around setting up a secure environment.

- Create and deploy a virtual machine using VMware 17 Pro to analyse the malware in.
    - Windows 10 machine for static and majority of dynamic analysis.
    - Kali Linux for memory analysis based on the windows 10 .VMEM file.
- Disconnect the virtual machines from the internet to isolate the malware from the internet.
- Disabling auto-run features to ensure malware would not run automatically.

These were the main components which secured the malware to the virtual machine. Once these steps were completed the procedures could start.

2.2 STATIC ANALYSIS

2.2.1 Basic static analysis

During this section, the goal was to discover basic attributes of the software. These include:

- Verify if the file is malicious.
- Find compile time.
- Break down all sections of the data.

- Verify whether the software is packed or not.
- View and analyse Imports which the software uses.

Using the File Hash application an MD5 hash was obtained, see Figure 1. Additionally, extra hashes were collected using the HashMyFiles application, these included SHA1, SHA-256, SHA-512 and more, see Figure 66. This revealed that the SHA-256 for the file matches that of the file name.

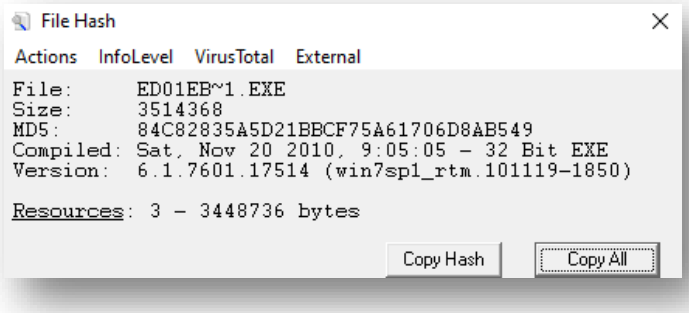

Figure 1 File Hash of software.

Using the MD5 hash and the online website Virustotal.com, the analyst was able to check and validate if the file was malicious. The website came back with several results, see Figure 2. For the full list of results please see Figure 67. This revealed that the file is in fact malware. The results indicated that the file is a type of ransomware and trojan. The most popular threat label was ransomware.wannacry/wannacryptor. This indicates that the application is the well-known malware called WannaCry which attack the NHS in 2017 and several other systems (NHS England, 2023). This is beneficial as several of online resources exist about this malware. This does not prove anything as it can only be used as an indication and the analysis must still take place.

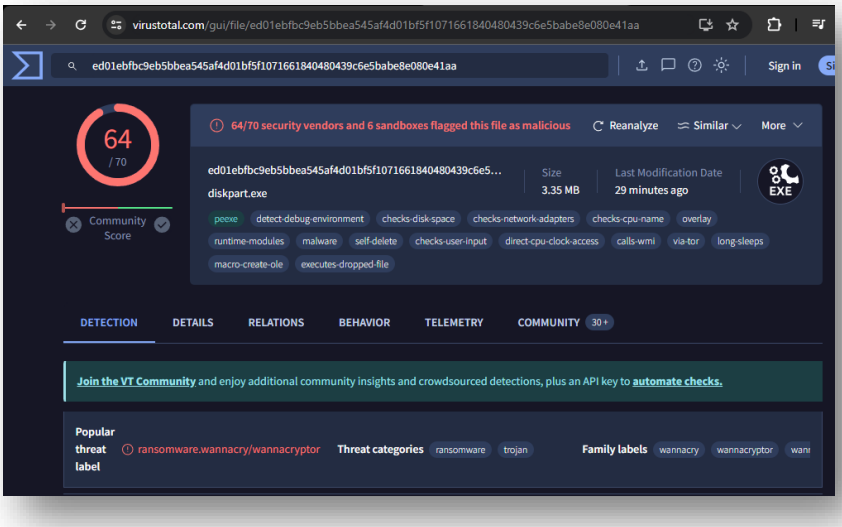

Figure 2 VirusTotal.com results.

The next step in basic static analysis is to find out when the malware was compiled. Using the tool PE Explorer and loading the malware into the tool revealed some intreating points. The compile time was found to be on 20/11/2010 at 09:05:05, see Figure 60. This was unusual as the application was not discovered until 2017. Research concluded that this time is accurate as experts have previous identified the same time (Kartone, 2019), it is expected that timestamp manipulation has taken place however, another possibility could be that the malware was compiled years before the deployment.

Additionally, the subsystem indicated that Win32 GUI is used, see Figure 60. This means that a graphical user interface is used for a windows machine. This will be verified during dynamic analysis.

To further analyse the malware another tool was used. DIE(Detect it easy) is a tool to help with static analysis. Loading the executable into the tool revealed that the application was compiled in C/C++ version 6.0 using microsoft linker version 6.0. Furthermore, all the data held within the archive seems to be packed via an encrypted zip file. This can be seen in Figure 3.

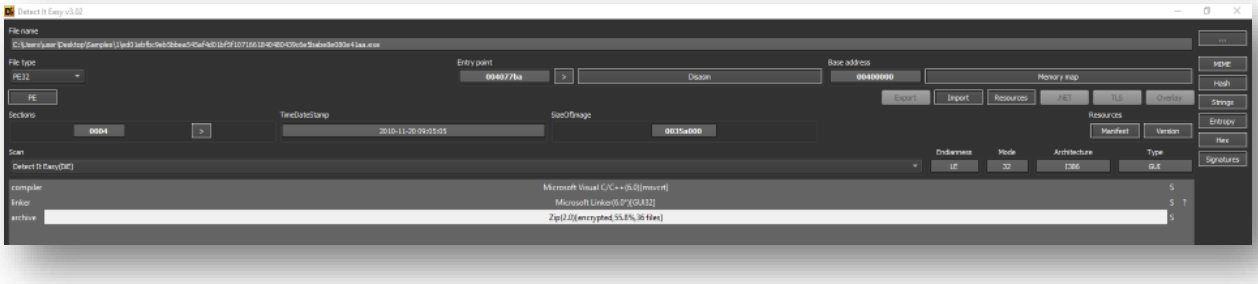

Figure 3 Main view from Detect It Easy.

Also Detect It Easy revealed that the application is broken down into 4 different sections, see Figure 4.

- .text
- .rdata
- .data
- .rsrc

While reviewing all of these sections the virtual size and the size of the raw data do not differ a large amount. This could indicate that no compression has taken place, but current analysis reveals that some resource data is packed using a zip file. Additionally, the application could be packed via encryption or other forms of obfuscation. The tool also revealed that the majority of the data is held within the .rsrc section.

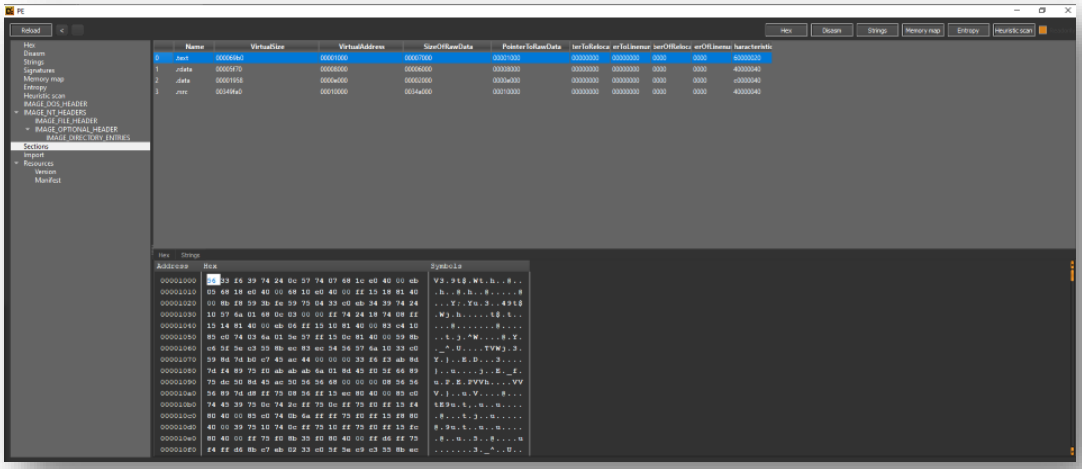

Figure 4 Discovered sections from Detect It Easy.

During the review of the entropy, 2 sections appeared to be packed according to DIE, see Figure 5. This could possibly be the zip file within the section. This could indicate that the executable will unpack and use data from the resource section for its processes while also avoiding detection. The zip file will be further analysed later in the report.

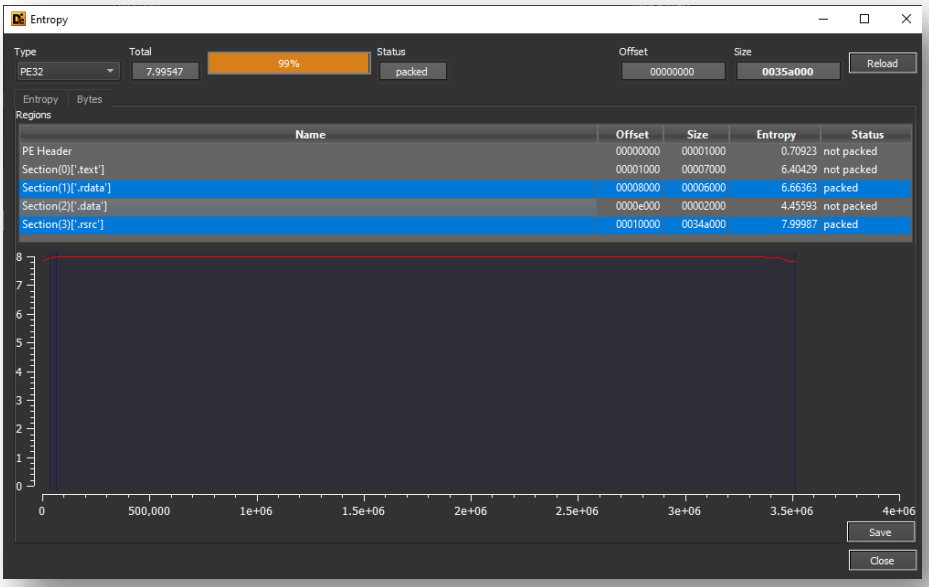

Figure 5 Entropy view from Detect It Easy.

The final test to identify packing is a string search on the application. Using the tool Strings.exe and the command shown below and piping the results to an output file returned thousands of strings but also revealed that majority of the strings are unreadable. This was predicted and hints at some sort of

encryption possible via the zip file. The output file is too large to be stored within this report, but strings of interest will be spoken about throughout the report. Furthermore, the output of the strings will be provided along with the report. This also hints at no use of packing software, for example UPX.exe. The analyst used upx.exe to perform a further test on whether the application is packed using this application. The results came back indicating a packed file but not packed via the application, see Figure 61.

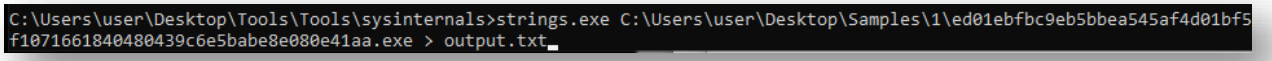

Figure 6 String search on malware.

Though this process the application appears to be packed but mostly the .rsrc section, this section could be used to hide the malicious functions and allow the main section of the data to utilize the resources but also hide from detection.

During the analysis of the resources, 3 subsections were found listed below, while reviewing these sections both RT_VERSION(16) and RT_MANIFEST(24) seem to be in clear text whereas XIA appears to be in an unreadable format according to the hex editor built into the DIE tool. This backs up the theory that the .rsrc section is packed.

- RT_VERSION(16)
- RT_MANIFEST(24)
- XIA

It was later discovered that the malware was packed with a zip file which then contained a further zip file. This will be seen during section 2.2.2

The final area of basic static analysis is identifying any imports which the executable uses. This was achieved using the tool, PE Explorer. 4 imports were identified below are the imports listed and explained.

- KERNAL32.dll
    - This library is used for communicating between hardware and software, additionally it is responsible for interacting with the OS (Windows) this includes:
        - Memory management.
        - Input/Output operations.
        - Process management.
        - Error handling.
- USER32.dll
    - This library is used for providing a useable GUI.
- ADVAPI32.dll
    - This library stands for Advance Windows 32 Base API. The dll file is responsible for
        - registry management.
        - service management.
        - device management.

- MSVCRT.dll
    - This library stands for Microsoft Visual C runtime. This is crucial for several functions listed below:
        - Standard C library functions.
        - Exception handling.
        - Thread management.
        - Dynamic memory management.
        - File operations and more.

These imports include a wide variety of functions; a full list can be seen in Appendix D – Identified Imports. This concludes basic static analysis. The next step was to analyse the code to further understand how the executable works.

2.2.2 Static code analysis

This section uses the tool Ghidra which is a reverse engineering tool developed by the National Security Agency (NSA). The goal for this section was to break down the overall application to gain a better understanding of how WannaCry operated. Another application which was considered was the IDA Pro application. However, the analyst chose Ghidra as it has an easy-to-use decompiler which meant the output was easier to analyse. This type of analysis can be highly technical and thus the report will attempt to simplify the contents.

Ghidra will decompile the executable from assembly and the analyst will then analyse the results and also change and alter the decompiled version to increase readability and reverse engineer functionality. This means each function had been analysed before the function was named.

The memory address 004077ba was identified to be the entry point by the tool, see Figure 7.

```
/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */

void entry(void)

{
  undefined4 *puVar1;
  uint nCmdShow;
  HMODULE hInstance;
  PWSTR pCmdLine;
  HINSTANCE hPrevInstance;
  char **local_74;
  _startupinfo local_70;
  int local_6c;
  char **local_68;
  int local_64;
  _STARTUPINFOA local_60;
  undefined *local_1c;
  void *pvStack_14;
  undefined *puStack_10;
  undefined *puStack_c;
  undefined4 local_8;
```

Figure 7 Executable entry point.

Further analysis was conducted, and this revealed that the entry point was the default entry point of any executable file. Generated by the MSVC linker version 6.0. At the bottom of this function another function was called, this was identified as the actual entry point. The analyst changed the signature to

the WinMain function taken from the windows website. This function takes 4 arguments according to the website (QuinnRadich, 2023), see Figure 8.

```
  local_6c = WinMain(hInstance,hPrevInstance,pCmdLine,nCmdShow);
                    /* WARNING: Subroutine does not return */
  exit(local_6c);
}
```

Figure 8 WinMain function call.

Upon examining this function several other functions are called so that the program can successfully work, see Figure 62 and Figure 63. The first thing the WannaCry does is call the function "GetModuleFileNameA" this function comes from the import found earlier which is called KERNAL32.dll. The function returns a path to the executable file. Below this function; another function is called, this is "randomstring_based_on_computer_name". This function generates a random string based on the computer name, see Figure 9 to view the functions content. This random string function calls another function from KERNAL32.dll to get the computer's name and save it as a variable. The variable is then used to seed the SRAND function which generates a random ASCII string and returns it back to the WinMain function. This string could be further used as a encryption key or a unique identifier.

```
GetComputerNameW(&local_19c,&computer_size);
local_8 = 0;
_Seed = 1;
sVar1 = wcslen(&local_19c);
if (sVar1 != 0) {
  pWVar6 = &local_19c;
  do {
    _Seed = _Seed * (ushort)*pWVar6;
    local_8 = local_8 + 1;
    pWVar6 = pWVar6 + 1;
    sVar1 = wcslen(&local_19c);
  } while (local_8 < sVar1);
}
srand(_Seed);
iVar3 = rand();
iVar7 = 0;
iVar4 = iVar3 % 8 + 8;
if (0 < iVar4) {
  do {
    iVar2 = rand();
    *(char *)(iVar7 + computer_name) = (char)(iVar2 % 0x1a) + 'a';
    iVar7 = iVar7 + 1;
  } while (iVar7 < iVar4);
}
for (; iVar7 < iVar3 % 8 + 0xb; iVar7 = iVar7 + 1) {
  iVar4 = rand();
  *(char *)(iVar7 + computer_name) = (char)(iVar4 % 10) + '0';
}
*(undefined *)(iVar7 + computer_name) = 0;
```

Figure 9 randomString_based_on_computer_name function.

After the function returns the string, the program then checks the "/I" argument. If the /I argument is there then it will execute the code within the if statement, see Figure 10.

```
                    /* if less than 1 arg quit */
argv = (int *)__p___argc();
if (*argv == 2) {
  _s_/i = &_Str2_0040f538;
  local_EAX_89 = (char ***)__p___argv();
  iVar6 = strcmp((*local_EAX_89)[1],(char *)_s_/i);
  if ((iVar6 == 0) &&
     (bVar1 = create_and_cwd_random_hidden_dir((wchar_t *)0x0), CONCAT31(extraout_var,bVar1) != 0)
     ) {
    CopyFileA(filename,s_tasksche.exe_0040f4d8,0);
    DVar2 = GetFileAttributesA(s_tasksche.exe_0040f4d8);
    if ((DVar2 != 0xffffffff) && (iVar6 = create_or_start_tasksche_service(), iVar6 != 0)) {
      return 0;
    }
  }
}
```

Figure 10 WinMain function checking arguments.

The program will then try and attempt to create a hidden directory with the name of the random string which was returned earlier. Then the executable will try and manipulate the attributes so the directory will have both "file_attribute_hidden and file_attribute_system". Then the directory is created within the $Windows\programData directory however if windows is not within the root directory the program will attempt to get the path to the Windows installation. There appear to be further attempts if these fail but if any of them are successful then the current working directory will be set to the path which is successful. In the virtual machine where dynamical analysis on the malware takes place the current working directory should be set to $\Windows\ProgramData.

After this step is completed, while still contained within the if statement, the program will copy itself to the tasksche.exe within the hidden directory that was created previously. The next function which is called is the "create_or_start_tasksche_service".

The functions content can be seen in Figure 11. Within this function the program attempts to create a service, however, if that fails the program calls a function "create_mutex_and_attempt_consitant" which will attempt to acquire a mutex and if the program fails, it will sleep for a minute before attempting again until the loop reaches its max iterations. If successful, the program will exit, see Figure 12 for functions content.

```
undefined4 create_or_start_tasksche_service(void)

{
  int iVar1;
  undefined4 *puVar2;
  CHAR local_20c;
  undefined4 local_20b;

  local_20c = DAT_0040f910;
  puVar2 = &local_20b;
  for (iVar1 = 0x81; iVar1 != 0; iVar1 = iVar1 + -1) {
    *puVar2 = 0;
    puVar2 = puVar2 + 1;
  }
  *(undefined2 *)puVar2 = 0;
  *(undefined *)((int)puVar2 + 2) = 0;
  GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&local_20c,(LPSTR *)0x0);
  iVar1 = create_service(&local_20c);
  if ((iVar1 != 0) && (iVar1 = create_mutex_and_attempt_consitant(0x3c), iVar1 != 0)) {
    return 1;
  }
  iVar1 = FUN_00401064(&local_20c,0,(LPDWORD)0x0);
  if ((iVar1 != 0) && (iVar1 = create_mutex_and_attempt_consitant(0x3c), iVar1 != 0)) {
    return 1;
  }
  return 0;
}
```

Figure 11 Create-or_start_tasksche_service function.

Below is a screenshot of the function “create_mutex_and_attempt_consitant”.

```
undefined4 __cdecl create_mutex_and_attempt_consitant(int param_1)

{
  HANDLE hObject;
  int iVar1;
  char local_68 [100];

  sprintf(local_68,(char *)&_Format_0040f4ac,s_Global\MsWinZonesCacheCounterMut_0040f4b4,0);
  iVar1 = 0;
  if (0 < param_1) {
    do {
      hObject = OpenMutexA(0x100000,1,local_68);
      if (hObject != (HANDLE)0x0) {
        CloseHandle(hObject);
        return 1;
      }
      Sleep(1000);
      iVar1 = iVar1 + 1;
    } while (iVar1 < param_1);
  }
  return 0;
}
```

Figure 12 create_mutex_and_attempt_consitant function.

After the if block was completed, the program sets the current working directory to the file containing the executable or within a hidden directory depending on whether the tasksche.exe was successfully created and if the /I argument exists. To see the other section of the WinMain function see Figure 13.

```
pcVar3 = strrchr(filename,0x5c);
if (pcVar3 != (char *)0x0) {
  pcVar3 = strrchr(filename,0x5c);
  *pcVar3 = '\0';
}
SetCurrentDirectoryA(filename);
set_or_query_reg_cwd(1);
                    /* takes the param and exstracts the zip file */
exstract_XIA((HMODULE)0x0,s_WNcry@2ol7_0040f52c);
FUN_00401e9e();
FUN_00401064(s_attrib_+h_._0040f520,0,(LPDWORD)0x0);
FUN_00401064(s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc,0,(LPDWORD)0x0);
iVar6 = FUN_0040170a();
if (iVar6 != 0) {
  FUN_004012fd();
  iVar6 = FUN_00401437((int)local_6e8,(LPCSTR)0x0,0,0);
  if (iVar6 != 0) {
    local_8 = 0;
    psVar4 = (short *)FUN_004014a6((int)local_6e8,s_t.wnry_0040f4f4,&local_8);
    if (((psVar4 != (short *)0x0) &&
        (argv = (int *)FUN_004021bd(psVar4,local_8), argv != (int *)0x0)) &&
       (pcVar5 = (code *)FUN_00402924(argv,s_TaskStart_0040f4e8), pcVar5 != (code *)0x0)) {
      (*pcVar5)(0,0);
    }
  }
  FUN_0040137a();
}
return 0;
}
```

Figure 13 Final section of the WinMain function.

The function “set_or_query_reg_cwd” is called which sets a registry key at HKLM\Software\WanaCrypt0r or if the attempt fails it will then create the key and set a value to the key. This function appears more complex than the rest and has proven difficult to analyse. The function can be seen in Figure 64 and Figure 65.

Another function was analysed, and this was enough to provide details to back up the rest of the analysis. The overall progress had been slow and there were several other functions to be processed which exceeded the scope of the analysis.

The other function analysed is called “exstract_XIA” this function takes an argument which appears to be an unusual string containing “WNcry@2ol7” Upon further analysis this revealed that this function used this string to extract resources from 2058. This was revealed earlier in Basic static analysis as the XIA section of the .rsrc section. This backs up the theory that XIA is packed using a zip file, see Figure 14.

```
hResInfo = FindResourceA(param1,(LPCSTR)2058,&DAT_0040f43c);
if (((hResInfo != (HRSRC)0x0) &&
    (hResData = LoadResource(param1,hResInfo), hResData != (HGLOBAL)0x0)) &&
   (pvVar1 = LockResource(hResData), pvVar1 != (LPVOID)0x0)) {
  DVar2 = SizeofResource(param1,hResInfo);
  piVar3 = (int *)FUN_004075ad(pvVar1,DVar2,pointer_to_pass);
  if (piVar3 != (int *)0x0) {
    local_130 = 0;
    puVar6 = local_12c;
    for (iVar5 = 0x4a; iVar5 != 0; iVar5 = iVar5 + -1) {
      *puVar6 = 0;
      puVar6 = puVar6 + 1;
    }
    FUN_004075c4(piVar3,-1,&local_130);
    iVar5 = local_130;
    pcVar7 = (char *)0x0;
    if (0 < local_130) {
      do {
        FUN_004075c4(piVar3,(int)pcVar7,&local_130);
        iVar4 = strcmp((char *)local_12c,(char *)u__0040e010);
        if ((iVar4 != 0) || (DVar2 = GetFileAttributesA((LPCSTR)local_12c), DVar2 == 0xffffffff))
        {
          FUN_0040763d(piVar3,pcVar7,(char *)local_12c);
        }
        pcVar7 = pcVar7 + 1;
      } while ((int)pcVar7 < iVar5);
    }
    FUN_00407656(piVar3);
    return 1;
  }
}
return 0;
```

Figure 14 Exstract_XIA function.

To test whether if the string is the password 7zip was used. The application was right-clicked, and the files were attempted to be extracted. As predicted a password was needed for the extraction, see Figure 15.

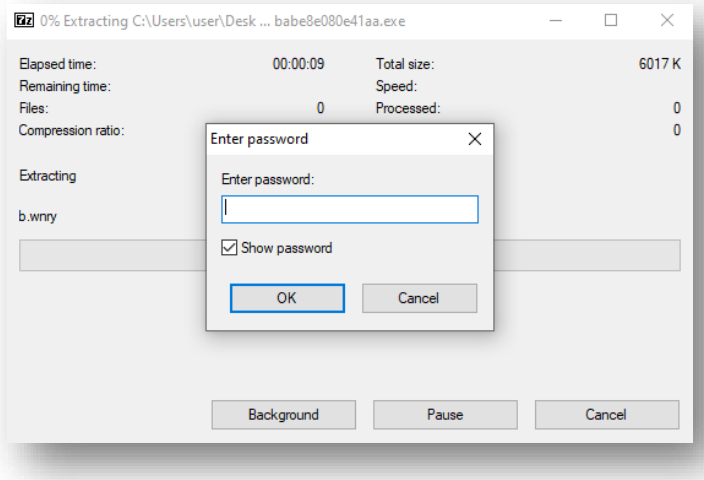

Figure 15 7zip attempted on the executable and password protected evidence.

The password “WNcry@2ol7” was entered into the password field and the extraction was successful. The files extracted can be seen below, Figure 16.

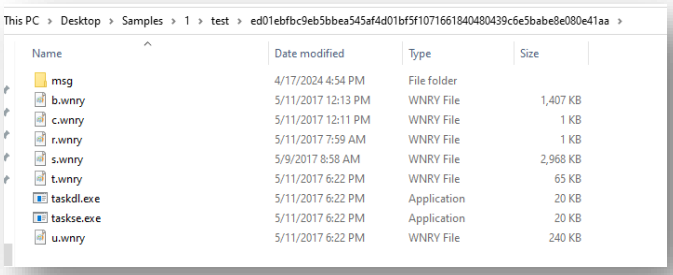

Figure 16 Original files extracted.

These files appear to be difficult to understand as the file extension does not provide any helpful information. The folder "msg" is easy to understand as within the file it has several files all with a different language. This hinted at the contents being a language interpreter for the WannaCry. Upon viewing the files each file contains the same amount of information just in different languages.

To further analyse the unknown files a tool called TrID was used to try and identify each file based on the magic number. This revealed an abundant number of helpful indications to each file. To improve the readability of the each file, each file was renamed to a more suitable name, see Figure 17.

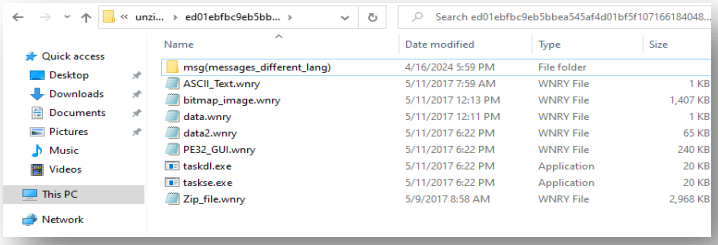

Figure 17 Renamed files which were extracted.

Each file was then opened in its respective application to successfully view the file. Below is each file explained.

- ASCII is a file which contains the help information to help users pay for their data, Figure 18.
- Bitmap is the file containing the image which is used as the desktop background, discussed during the dynamic analysis Figure 19.
- Data.wnry is the file containing tor URL's and a URL to the tor download folder, the URL is outdated but the start of the URL still works, see Figure 25. This could be used to direct payments to the website contained within the dark web, Figure 20.
- Data2.wnry is unreadable and contains no useful information.
- PE32 GUI is a file containing information possibly used for the graphical user interface which is used then the malware is executed Figure 21. This will be further analysed as it is a PE32 file which indicates an executable.
- Taskdl.exe has a description which indicates that it is used for SQL configuration Figure 22.

- Taskse.exe has a description which indicates that it is used to send and receive signals Figure 23.

7zip was used on the final file (Zip_file.wncy) this successfully extracted two folders which appears to be related to Tor. The Tor folder contains several files mostly dynamically linked libraries. However, another executable file was found called tor.exe. This attempts to connect to the dark web. This could be used for sending payments or for establishing a connection. The final folder which was extracted was the Data folder, the folder was empty and is expected to be populated with data retrieved from the Tor browser upon successful connection, see Figure 24.

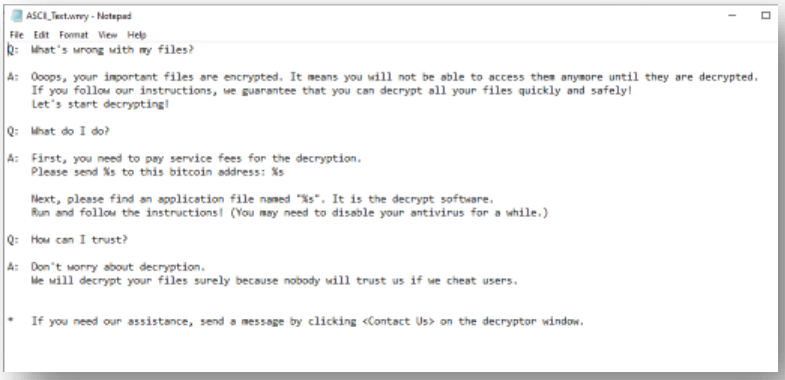

Figure 18 ASCII file containing information and help which is displayed once the malware is executed.

Below is the screenshot of the WannaCry desktop background.

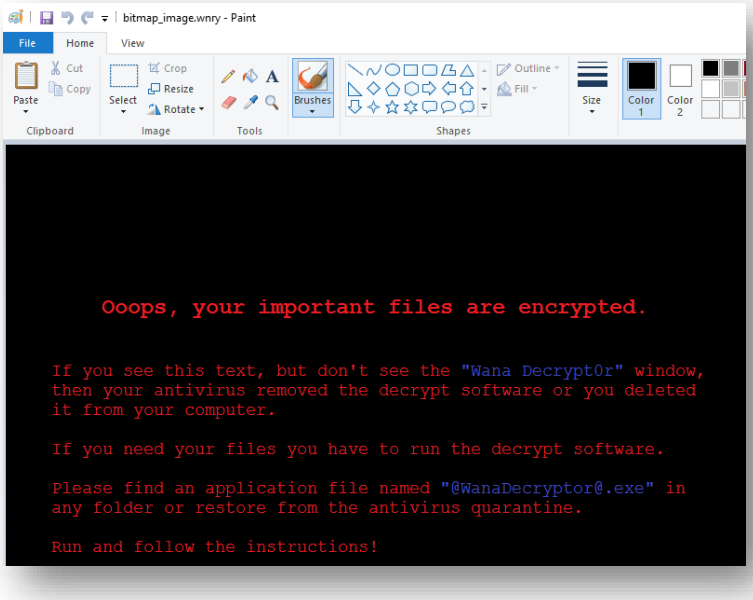

Figure 19 Bitmap file containing an image which sets the desktop background.

Below is a screenshot of the text file found, containing the onion addresses.

Figure 20 Data file containing tor URL's and the URL to download tor zip file.

Below is a file containing unreadable content relating to PE32_GUI.

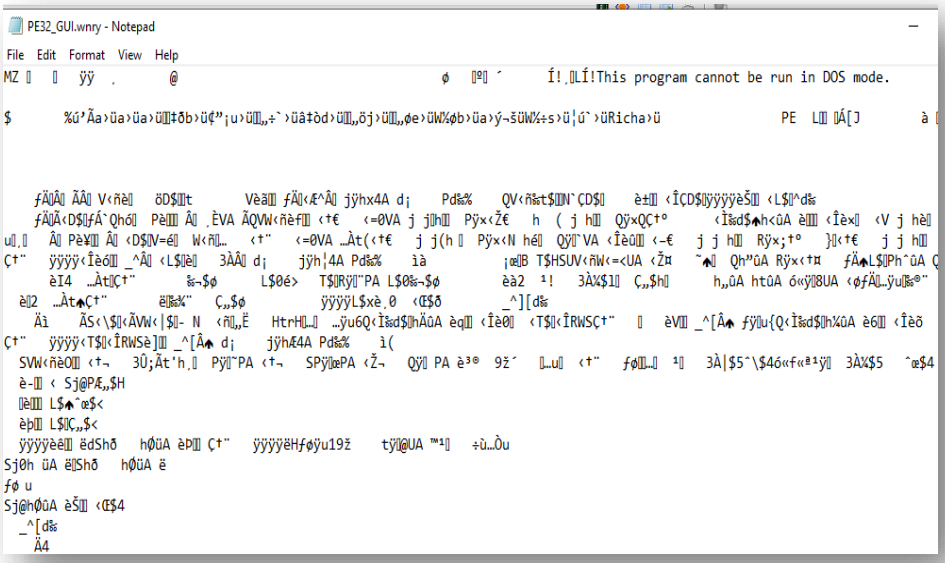

Figure 21 PE32 GUI file contain expected GUI which pop's up during the execution of malware.

Below is a screenshot of the description for taskdl.exe.

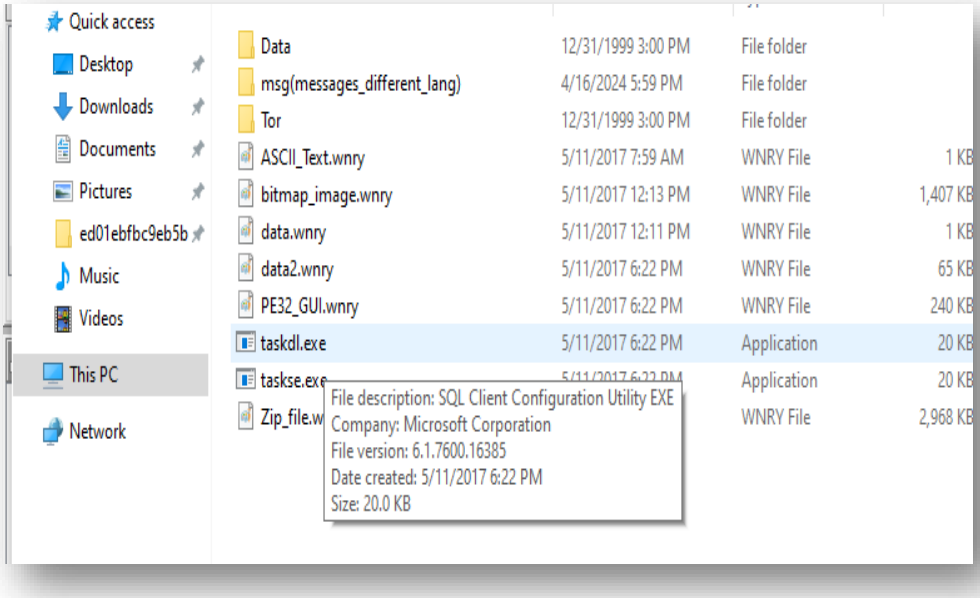

Figure 22 Taskdl.exe description.

Below is a screenshot of the descripting relating to taskse.exe.

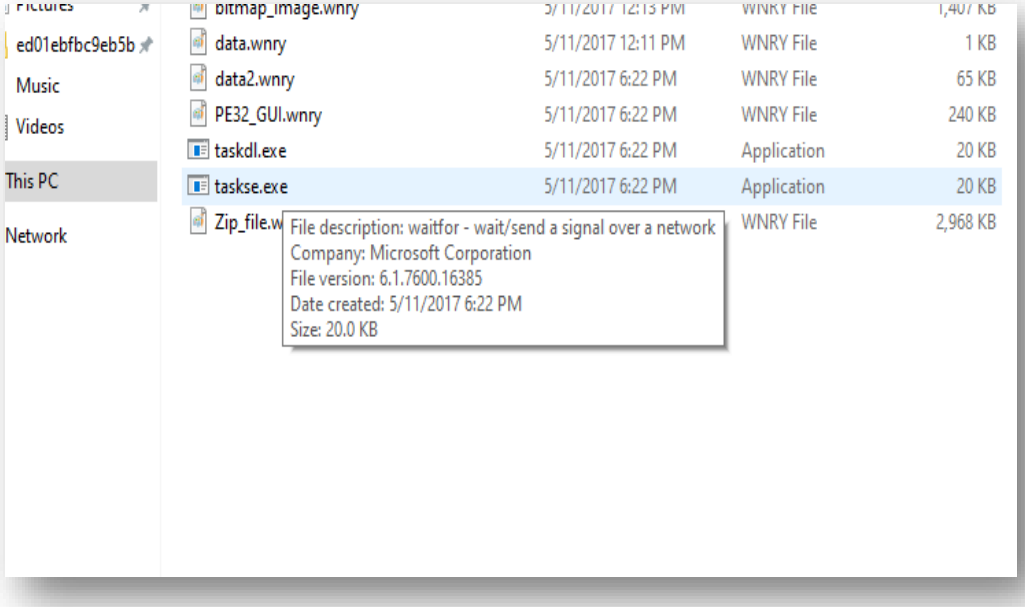

Figure 23 Taskse.exe description.

Below is a screenshot showing the inside of the Tor folder and the Data folder relating to Tor.

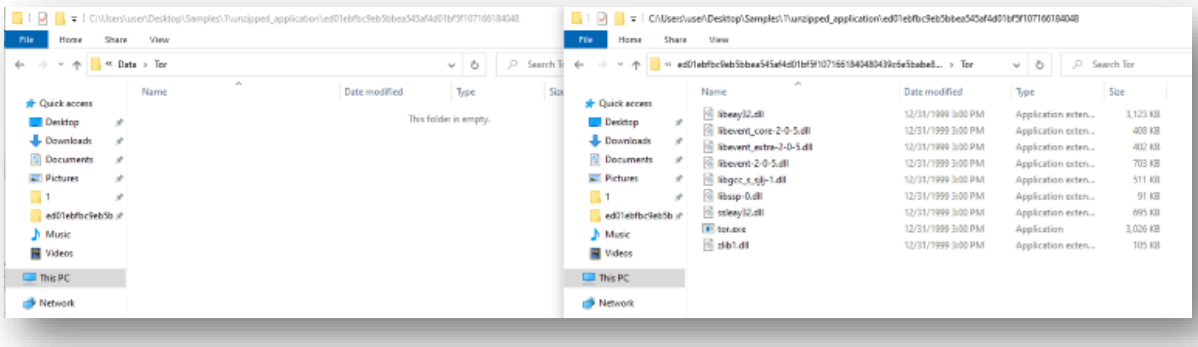

Figure 24 Extracted folders from Zip_file.wnry.

Below is a screenshot of the URL found which reveals it is still active apart from the version URL extension.

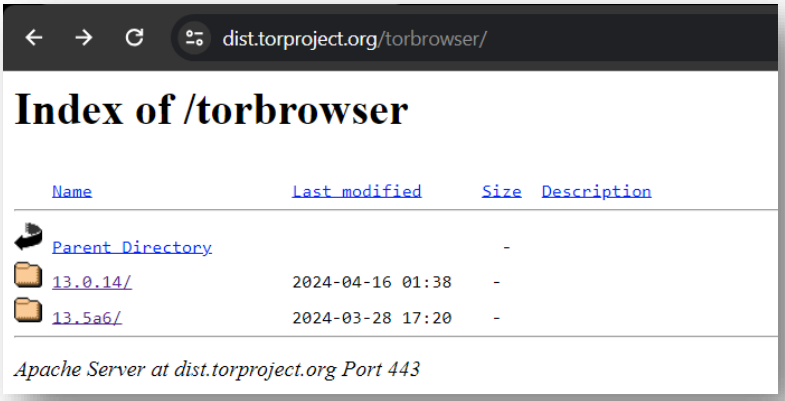

Figure 25 Browser of the tor zip URL found in c.wnry.

PE32 GUI.wnry was loaded into DIE(Detect it easy) and this reveals that it contains attributes of an executable, see Figure 26.

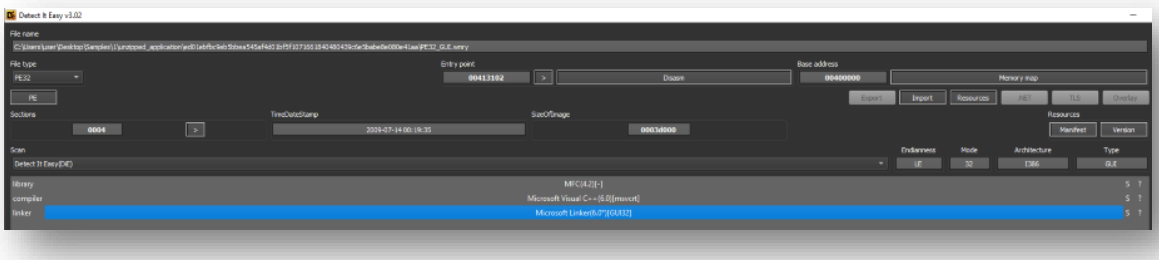

Figure 26 DIE results on PE32 GUI.wnry

The analyst changed the extension to a .exe and ran it. This could be argued that this is dynamic analysis. However, as this is a subsection of the malware, the executable is more likely to be legitimate software. This was still conducted within a secure environment with a safe snapshot for security. Once the extension was changed the logo beside the file in file explorer changed to an image which looks like a handshake. Once the software was executed a GUI was displayed as predicted, the GUI is the one which is used during the WannaCry attack but without any context. This will be shown in the Dynamic analysis. See Figure 27.

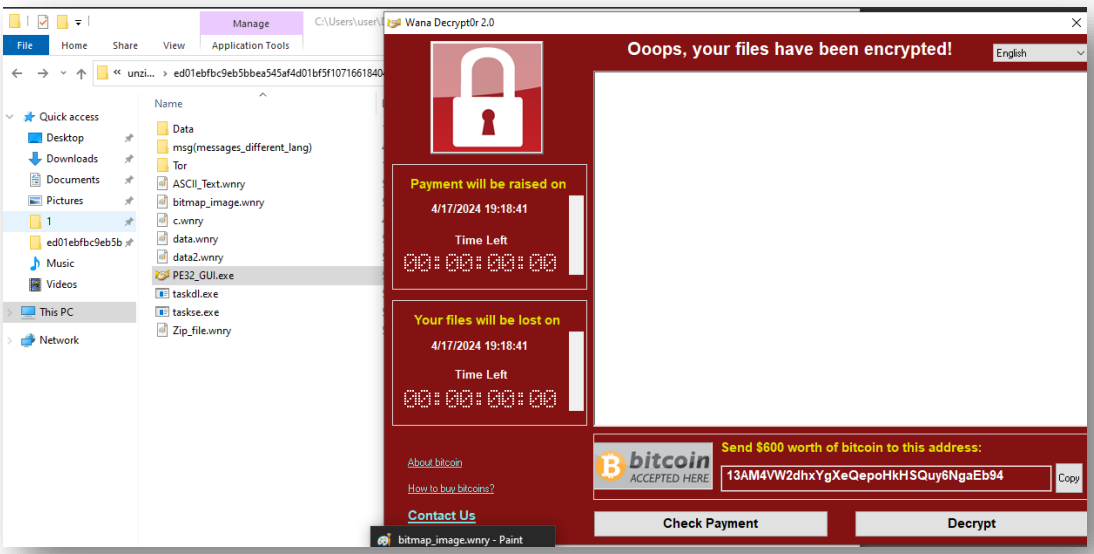

Figure 27 PE32_GUI executable results.

A varied amount of evidence was found including proof that a zip file exists, which indicates partial packing of the software. Additionally, a further zip file was found. This revealed that partial packing exists, and nested packing also exists.

2.2.3 Further Static Code Analysis.

After the analyst conducted dynamic analysis, the analyst then proceeded with further static code analysis. This investigation was extensive, so this section will be summary of what was found to further advance the report.

This section is covering the final area of the WinMain function, see Figure 28.

```
FUN_00401e9e();
FUN_00401064(s_attrib_+h_._0040f520,0,0);
FUN_00401064(s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc,0,0);
iVar5 = FUN_0040170a();
if (iVar5 != 0) {
  FUN_004012fd();
  iVar5 = FUN_00401437(0,0,0);
  if (iVar5 != 0) {
    local_8 = 0;
    iVar5 = FUN_004014a6(s_t.wnry_0040f4f4,&local_8);
    if (((iVar5 != 0) && (iVar5 = FUN_004021bd(iVar5,local_8), iVar5 != 0)) &&
       (pcVar4 = (code *)FUN_00402924(iVar5,s_TaskStart_0040f4e8), pcVar4 != (code *)0x0)) {
      (*pcVar4)(0,0);
    }
  }
  FUN_0040137a();
}
return 0;
```

Figure 28 Further static code analysis overview.

2.2.3.1 FUN_00401e9e

This function appears to contain several bitcoin addresses. This would be used to help pay the ransomware. The number generates a random number which is used to select the bitcoin address. Additionally, another function is called within this function. The function "FUN_00401000 reads and writes to a file called s_c.wnry_0040e010. Earlier, the file "c.wnry" was identified to have several onion addresses. The function then returns a variable to the function "FUN_00401e9e".

In summary this function seems to pick a random string relating to a bitcoin address and then attempts to connect to 1 of 5 onion addresses. This could be related to selecting the bitcoin address which is displayed within the GUI to send bitcoin.

2.2.3.2 FUN_00401064

Upon an in-depth review of these two functions, It appears that one changes permissions on a directory, possibly to perform some obfuscation. The other appears to set elevated permissions on a user based on the parameters passed in.

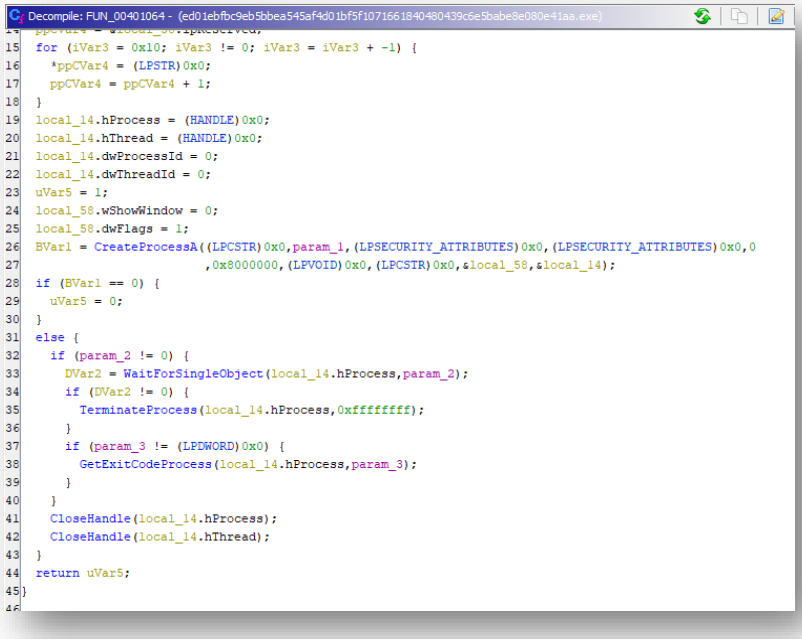

Figure 29 Contents of FUN_00401064.

2.2.3.3 FUN_004012fd

This function is used to increase overall resilience, calling other functions to create exceptions and improve error handling, See Figure 30.

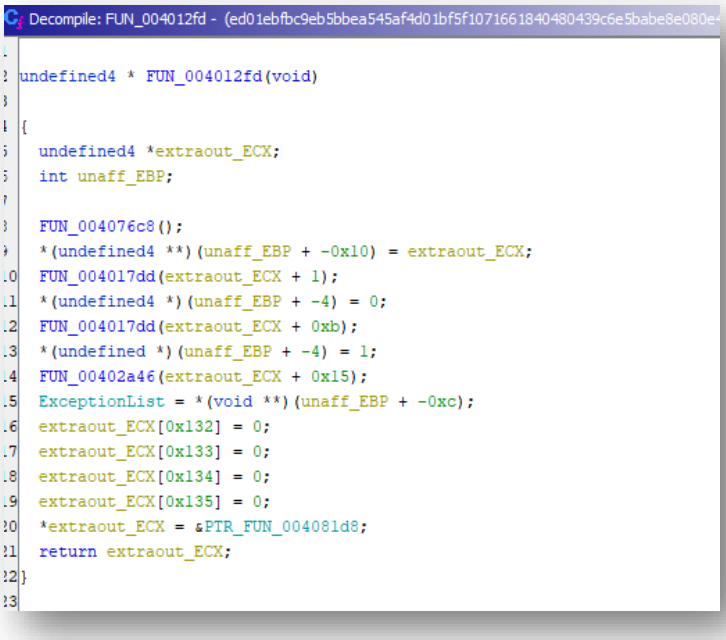

Figure 30 Static code analysis contents of FUN_004012fd

2.2.3.4 FUN_0040170a

This function loads library addresses relating to kernal32.dll. The function retrieves a specific address for each function and returns a true or false value, depending on whether the process was successful, see Figure 31.

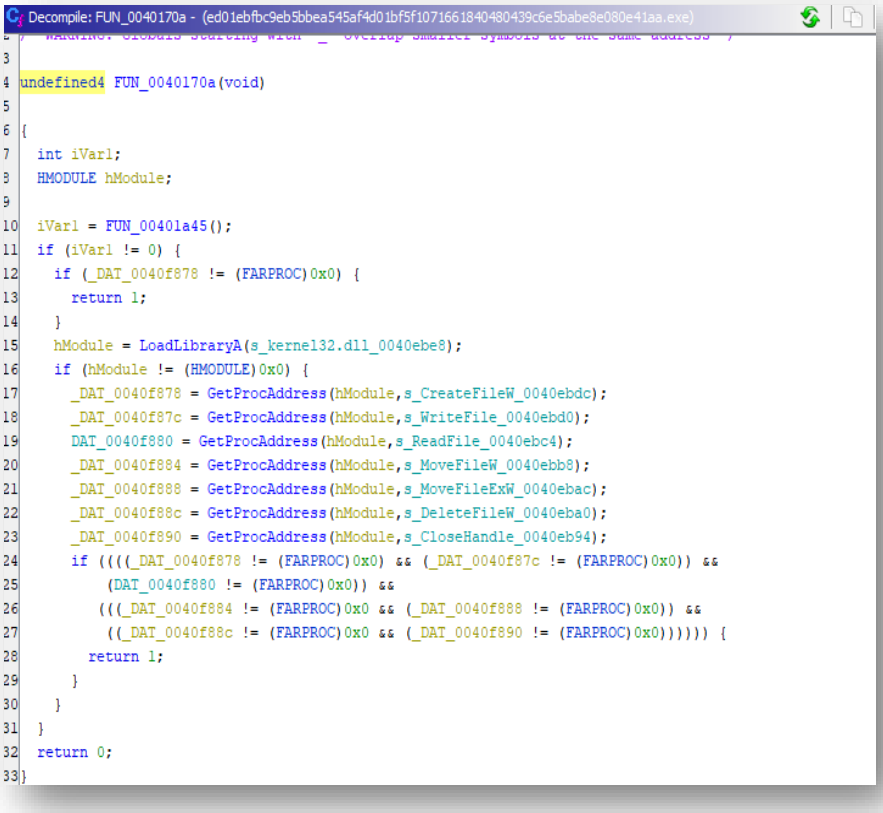

Figure 31 Contents of FUN_0040170a.

2.2.3.5 FUN_00401a45

This function creates pointers for different operations by loading a library specifically, kernal32.dll and retrieving the address of a specific function. This function returns a 1 or 0 result, depending on whether the process was successfully completed similar to the previous function.

Figure 32 Contents of FUN_00401a45

2.2.3.6 FUN_00401437

This function appears to manipulate memory allocation returning true if the memory allocation was successful otherwise, it will return false. Upon further analysis, this function calls FUN_00401862 which performs operations and data manipulation. This could be a component in the encryption of WannaCry. This could also be a component to check whether the file is encrypted to avoid re-encryption, see Figure 33.

Figure 33 Contents of FUN_00401437.

2.2.3.7 FUN_004014a6

This function decrypts the content of t.wnry which was unreadable during the previous analysis, see section 2.2.1. Additionally, this function calls several functions inside to help perform data processing and error handling. Finally, one of the function calls appears to handle a parameter passed in. The function then proceeds to perform a complex data transformation, likely encrypting the file passed in. the functions called within this function are,

- FUN_004019e1.
- FUN_00402a76.
- FUN_00403a77.

2.2.3.8 Summary

The above functions have all been analysed they all fit the profile identified in dynamic analysis however, detailed analysis should be undertaken, As the analyst went deeper into the rabbit hole it was clear that this started to go beyond the scope. See Future Work.

2.3 DYNAMIC ANALYSIS

2.3.1 Basic dynamic analysis

During this section the report will examine the WannaCry malware dynamically, This technique involves running the application and using a series of tools to piece together how the application runs. It is pivotal that section 2.1.4 is correctly done because, if the setup is wrong this could damage the

computer which the analysis was performed on. The tools which were used are discussed during section 2.1.2.

This process is quite extensive, and multiple data was passed around the device, majority of the screenshots for this section can be seen in Appendix A Dynamic Analysis. During basic dynamic analysis 4 areas will be examined these include observing the malware, process monitoring, registry analysis and network analysis.

2.3.1.1 Observing the malware

This section will iterate over what the analyst observed during the execution of the malware.

Once the malware was executed by the analyst the CPU usage started to increase significantly, suggesting a considerable number of threads and processing power was being used. Extra files on the desktop were also noticed including a new executable file called @WanaDecryptor@.exe This file automatically opened up. Upon viewing it was obvious that this was the PE32 GUI which was discovered earlier during static code analysis. Upon viewing the PE32 GUI executable several further functions were available. These included,

- Contact us link to send a message.
- Decrypt function which will decrypt 10 random files to show that the files are retrievable.
- Check payment which will connect to a server to check if the payment was received before decrypting all the files.
- Further links which will give information to the user to help recover their files.

Additionally, the desktop background was changed to a new image. This image had a background colour which was black and red text which informed the user about their files being encrypted. Finally, all the encrypted files start with the text WANACRY! In a hex editor.

During the start-up of the malware a windows pop-up appeared asking the user to allow the process to continue, see Figure 34. During a further analysis of this pop-up the software appeared to be malicious and further details were identified, These are listed below,

- “Wmic shadowcopy” delete is a command which deletes all backups which windows could use to recovery the system.
- “Bcdedit /set {default} bootstatuspolicy ignorerealfailsures” & “bcdedit /set {default} recoveryenabled no”, this section of the pop-up suggests that the boot configuration is modified to ignore any failures during start-up which are caused by the malware. This will prevent windows attempting to repair itself.
- “Wbadmin delete catalog -quite”, forces windows to delete the backup and the quite flag is used to prevent further confirmation.

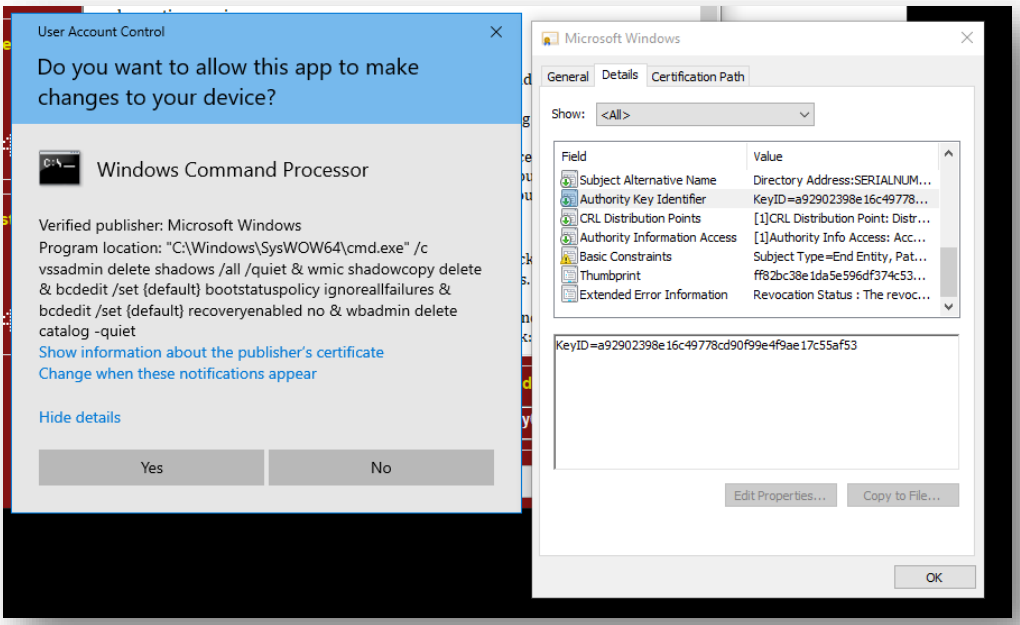

Figure 34 Windows account control pop-up.

2.3.1.2 Process monitoring

Using Process Monitor and Process Explorer the analyst was able to analyse different processes which the malware created. For Process Explorer filters were used to analyse only the processes which were utilized by the malware. Several processes started these included,

- The actual application, which was manipulating the disk partition, this is due to the encrypting which the application was doing.
- @WanaDecryptor@.exe, is a subprocess of the application, this is responsible for operating the GUI which alerts the users.
- Taskhsvc.exe is a further subprocess of the @WanaDecryptor@.exe and this could be connected to the taskdll.exe or taskse.exe.

Another process was found in correlation to the original application. The name of the process is Isass.exe which is a local security process. This suggests that the application could use this process to bypass authentication, change security policies to propagate and also avoid security auditing, see Figure 35.

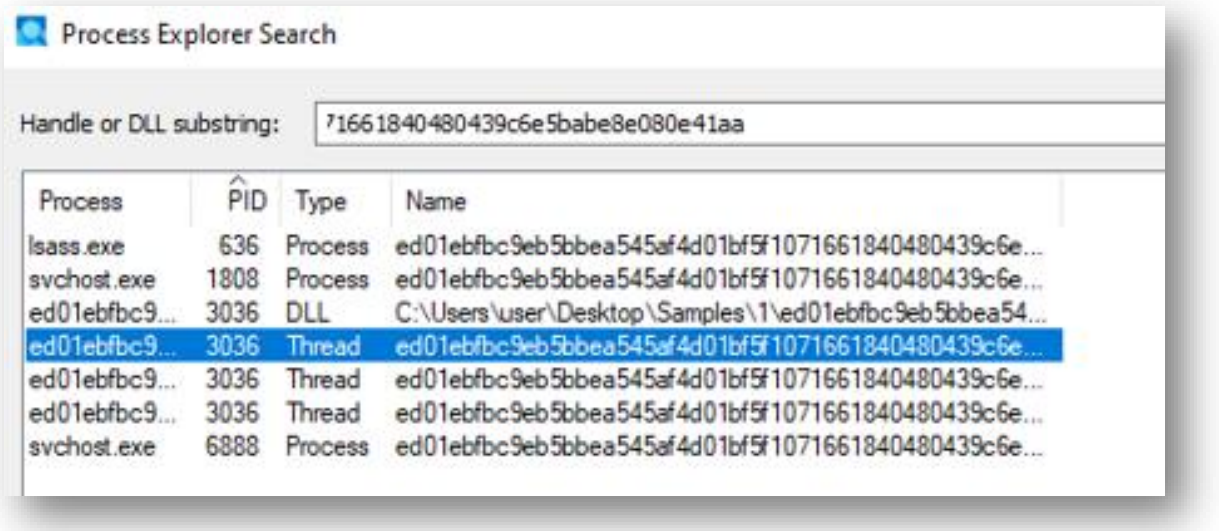

Figure 35 Proof of lsass.exe with the same as the application.

The analyst further analysed the processes and their activity using Process monitor. Numerous amounts of data were sniffed, over 2 million events happened before the collection of data was stopped. Using filters to filter out information and follow their activity. This was done by following PID(process id) and TID(Thread id). Additionally, the filter could be set to filter out the operation that was done.

The analyst noticed that several threads were created to accelerate the application and perform several tasks throughout the execution. These tasks are shown below, see Appendix A Dynamic Analysis for screen shots.

- Processes
  - Process created was found, it appears to create the processes taskse.exe and the process which executes graphical user interface.
  - Thread Create, Thread Exit.
  - Process Start which was identified to start a particular process.
  - Load Image was identified in connection with WOW32 (windows on windows 32 bit).
- File manipulation
  - CreateFile was identified, The file created was in the AppData\Roaming\Microsoft\Protect\S-1-5-21-2169232433-3398496680-935370409-1000\Preferred. Further files were created.
  - CloseFile was found throughout the application which suggests files were check and possibly manipulated before being closed. Most likely to encrypt the contents of each file.
  - QueryNameInfoFile was found throughout the application; these backs up the theory during static code analysis where the file and taskdll.exe were checked before the application proceeded.
- Registry editing.
  - RegSetInfoKey this is used to set class information for a registry key.
  - RegOpenKey was used to open the key locations

- RegCloseKey was used to close keys once their purpose was completed.
- RegQueryValue was identified several times this was used to identify the value of the reg key. These locations varied significantly. See Appendix A Dynamic Analysis.

These tasks were identified and directly contributed to the execution of WannaCry, Additionally, some mutexes were found which suggests the process was trying to edit the same resource. WannaCry might use the mutex to avoid running multiple instances of itself on the same resource or system.

2.3.1.3 Registry analysis

During registry analysis the tool used to capture changes was Regshot. This tool was used before the execution and after the malware ran for around minutes the 2nd shot was taken. Comparing the two results and outputting to a file which explained different changes to the registry. The previous section discusses the registry edits, this section will clarify some of the changes made.

Regshot identified 3 keys which were deleted, The first appears to be related to Google updates and could contain information about any updates whether in progress or successfully completed. The second appears to be related to information about Internet Explorers history. This indicates that the history was deleted after Tor was downloaded onto the malware. The WannaCry version that was analysed appears to have a pre-installed version of WannaCry as it is unable to download the Tor browser due to no internet connection. In addition, the final key which was changed appears to be related to cached website pages, see Figure 36.

Figure 36 Regshot Keys deleted.

85 keys were added, as this is an ample amount; the results can be seen in Figure 48 and Figure 49. To summarise, Several keys were changed for a specific user as the user ID was the same throughout the output file. This appears to be related to permissions, internet explorer and software keys.

14 values were deleted, these appear to be connected with internet history including user history, see Figure 37.

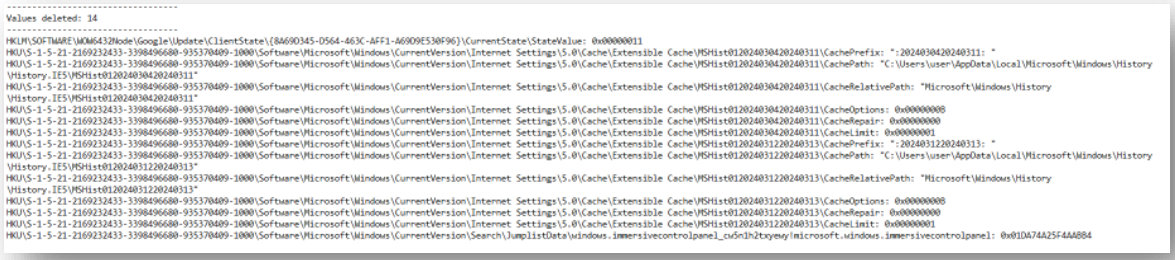

Figure 37 Regshot values deleted.

Regshot identified 221 values to be added. These do not all appear to be related to WannaCry. As majority of them relate to process explorer, VirusTotal, Opacity and more. Some values could relate to the malware but no evidence to support WannaCry's main functionality can be seen.

Finally, Regshot identified 74 values which were modified. Similar to the previous section not all of the entries relate to WannaCry. Below are some entries of interest,

- ThrottleCatche\UserID\ThrottleStarterTime appears to be mechanism which stores a specific timestamp.
- LastRefreshAttempted stores a value of the last time the program was refreshed. This could relate to the refresh attempt discussed during the static code analysis, additionally the refresh could also be the cause of the GUI consistently being called.
- DynamicInfo held within the HKLM\Software hive appears to relate to task scheduling. This could be use by WannaCry to allow tasks to be scheduled.
- HKLM\Software had another entry called RefCount. This indicates that any changes to the user's profile could be referenced for further use by the malware.
- HKLM\System has an entry called bam\status\... This is most likely used to allow the malware to propagate throughout the system. Changes to BITS(background intelligent transfer service) could indicate an attempt to alter its behaviour.
- The HKU hive has a modified values in the Desktop\WallPaper which indicates the changes to the wallpaper as discovered during dynamic analysis.
- The HKU hive also had a modified value at Winlogon\PUUActive. This indicates that changes were made relating to user's authentication and/or login settings.

Additionally, further analysis needs to be conducted to fully examine the changed content of the registry, see Future Work. Furthermore, the longer the application runs the more changes will be made. In total 397 changes were made in a short period of time. A file with the output of Regshot will be provided alongside the report. This concludes registry analysis.

2.3.1.4 Network analysis

During network analysis three tools were used to simulate a network and capture the packets being sent. These were Fakenet, ApateDNS and Wireshark.

Fakenet and ApateDNS were used to simulate network traffic whereas most of the analysis was conducted using Wireshark.

Fakenet identified several connection attempts. From process attempting to connect to several ports, the port of interest was 9050 as this appeared most often. Another point of interest was the process taskhsvc.exe which appears to try and establish a connection.

The main process WanDecryptor@.exe which is responsible for decrypting the files and connecting to the Tor browser, connects to an IP on port 9050. These can be seen in Figure 50 and Figure 51. The analyst analysed this further using Wireshark.

At first glance, the results are large and difficult to analyse, see Figure 52. To further analyse this a filter was applied see Figure 53. This filtered the results down to packets which were trying to connect to an onion browser. This revealed that the process was attempted to connect to an onion browser. Upon further analysis it was clear that 5 addresses were attempted, these were also found during static analysis. Figure 54 Figure 55 Figure 56 Figure 57 Figure 58 shows all the onion browsers within the Wireshark hex editor.

The full results can be seen in Figure 59. This concludes network analysis.

2.3.2 Memory dynamics analysis.

To perform memory analysis a .vmem file was needed. This was captured during dynamic analysis. Additionally, the tool volatility (Windows version) was used to perform the memory analysis.

Several commands were called and outputted to a file. However, majority of the data only back-up information from previous sections and no new data was retrieved. An interesting piece of information found was taskhsvc.exe attempting to connect to Tor on a specific IP on port 443, see Figure 74. Additionally, Figure 77 reveals all the .dll which are used during the analysis which were still on the memory. This concludes memory analysis, all the results from this investigation can be seen in Appendix E – Memory analysis. The Discussion section will analyse all the results found during the full investigation.

3 DISCUSSION

3.1 GENERAL DISCUSSION

This malware analysis conducted revealed several features of the malware including processes, network functionality and the impact which the malware has on the computer system. This report met the aims set out during section 1.2 accordingly. More extensive analysis could be performed but this stretches beyond the scope of the report. The methodology set out before the procedures was followed to give a full comprehensive analysis. The SAMA methodology would be the primary methodology if the analyst did not cater the methodology to the investigation, as it was more in-depth and suitable. Finally, the report had successfully described each step taken in an easy-to-follow systematic approach.

Several indicators of compromise (IOC's) were found. These are listed below,

- Hash of a malicious file, identified by Virus Total.
- Changes of registry entries.
- Out bound network traffic attempting to connect to an onion browser.
- Behaviour anomalies, including WanaDecrypt@.exe.
- Changes to files including the encryption of personal files.

These list the main IOC's which indicate a certain type of malware. This report could be further used to help create software to combat these and additionally, this helped meet the overall purpose of a malware analysis.

Finally, The WannaCry malware that was analysed was different to the original WannaCry; as it did not contain a kill switch. This means that there was no way of preventing the malware from executing, as with the original version the kill switch attempted to access a URL. If the request was successful, the WannaCry would not proceed any further with the encryption and propagation phase.

3.1.1 Conclusion

In conclusion, the work conducted revealed the software investigated was malicious, and it falls within the category of Ransomware. This is obvious to deduce as the software encrypts the user's files, propagates throughout the system, and demands payment from the victim. Through the use of cryptocurrency via Tor. Additionally, VirusTotal.com identified a Trojan signature which could be argued as the malware hides itself within a directory named as a random string.

3.1.2 Countermeasures

This malware is quite outdated and won't be able to infect the current Window 11, additionally this variant of WannaCry won't be able to infect any Linux or MacOS. However, for older versions of Windows OS, there are some techniques that can be deployed to increase overall protection, these are;

- Regular updates to software.
- Enables firewalls and correct configuration.
- Implement strong access controls.
- Back-up data regularly to an external server.

3.2 FUTURE WORK

During this report several other areas were still un-investigated, this is due to the restriction of time and lack of advance knowledge. This section will identify several areas which could undergo more analysis, see below.

- Further Static code analysis.
    - During static code analysis it was clear that the amount of time to fully reverse engineer WannaCry was beyond the scope. To analyse the malware further functions and a more detailed breakdown of the malware could be conducted as this could be used to create more secure defences and understand how to recreate the full application.
- Reverse engineer the decryption key.
    - It was identified during research that the decryption key is possible to reverse engineer the decryption key. This would be highly beneficial as it would be used to prevent any other WannaCry attacks.
    - A tool which could be used is “Wanakiwi”, the issue found with this tool is that it only works on Windows XP and 7. This meant that the tool would not work on the environment which the analyst worked on.
- Attempt to decrypt files, could use cyber chef for any strings which are encoded and are unreadable.
    - This could be beneficial as there was still a “t.wnry” file which was encrypted and unreadable. Although the static code analysis breaks down the functionality the contents are still unavailable, this could help identify the content and possible advance the investigation.

REFERENCES

Arctic Wolf (2021). *The 8 Most Common Types of Malware*. [online] Arctic Wolf. Available at: https://arcticwolf.com/resources/blog/8-types-of-malware/ [Accessed 10 Apr. 2024].

AV-TEST (2019). *Malware Statistics & Trends Report | AV-TEST*. [online] Av-test.org. Available at: https://www.av-test.org/en/statistics/malware/ [Accessed 11 Apr. 2024].

AVG (2015). *What is Malware? How Malware Works & How to Remove it*. [online] Avg.com. Available at: https://www.avg.com/en/signal/what-is-malware [Accessed 10 Apr. 2024].

Bermejo Higuera, J., Abad Aramburu, C., Bermejo Higuera, J.-R., Sicilia Urban, M.A. and Sicilia Montalvo, J.A. (2020). Systematic Approach to Malware Analysis (SAMA). *Applied Sciences*, 10(4), p.1360. doi:https://doi.org/10.3390/app10041360.

First.org (n.d.). *Malware Analysis Framework v1.0*. [online] FIRST — Forum of Incident Response and Security Teams. Available at: https://www.first.org/global/sigs/malware/ma-framework/ [Accessed 23 Apr. 2024].

Fortinet (2024). *Recent Ransomware Payments and Settlements*. [online] Fortinet. Available at: https://www.fortinet.com/uk/resources/cyberglossary/recent-ransomware-settlements [Accessed 10 Apr. 2024].

Fox, N. (2022). *How to Use Volatility for Memory Forensics and Analysis | Varonis*. [online] www.varonis.com. Available at: https://www.varonis.com/blog/how-to-use-volatility [Accessed 23 Apr. 2024].

Kartone (2019). *Reverse engineering of a WannaCry sample*. [online] Kartone Infosec Blog. Available at: https://blog.kartone.ninja/2019/05/23/malware-analysis-a-wannacry-sample-found-in-the-wild/ [Accessed 14 Apr. 2024].

McAfee (2023). *What is malware and why do cybercriminals use malware?* [online] McAfee. Available at: https://www.mcafee.com/en-gb/antivirus/malware.html [Accessed 10 Apr. 2024].

NHS England (2023). *NHS England» NHS England Business continuity management toolkit case study: WannaCry attack*. [online] www.england.nhs.uk. Available at: https://www.england.nhs.uk/long-read/case-study-wannacry-attack/ [Accessed 14 Apr. 2024].

QuinnRadich (2023). *The WinMain application entry point - Win32 apps*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/windows/win32/learnwin32/winmain--the-application-entry-point [Accessed 16 Apr. 2024].

Radware (2024). *Malware Development - A History - Radware Application and Network Security*. [online] www.radware.com. Available at: https://www.radware.com/resources/malware_timeline.aspx/#:~:text=The%20%22Creeper%20Virus%22%20was%20created [Accessed 10 Apr. 2024].

Rouse, M. (2016). *What is Kernel32.dll? - Definition from Techopedia*. [online] Techopedia.com. Available at: https://www.techopedia.com/definition/3379/kernel32dll [Accessed 15 Apr. 2024].

Singer, P.W. (2015). Stuxnet and Its Hidden Lessons on the Ethics of Cyberweapons. *Case Western Reserve Journal of International Law*, [online] 47(1). Available at: https://scholarlycommons.law.case.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1009&context=jil [Accessed 2 May 2024].

VirusTotal (2024). *VirusTotal*. [online] www.virustotal.com. Available at: https://www.virustotal.com/gui/file/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa [Accessed 14 Apr. 2024].

APPENDICES

APPENDIX A DYNAMIC ANALYSIS

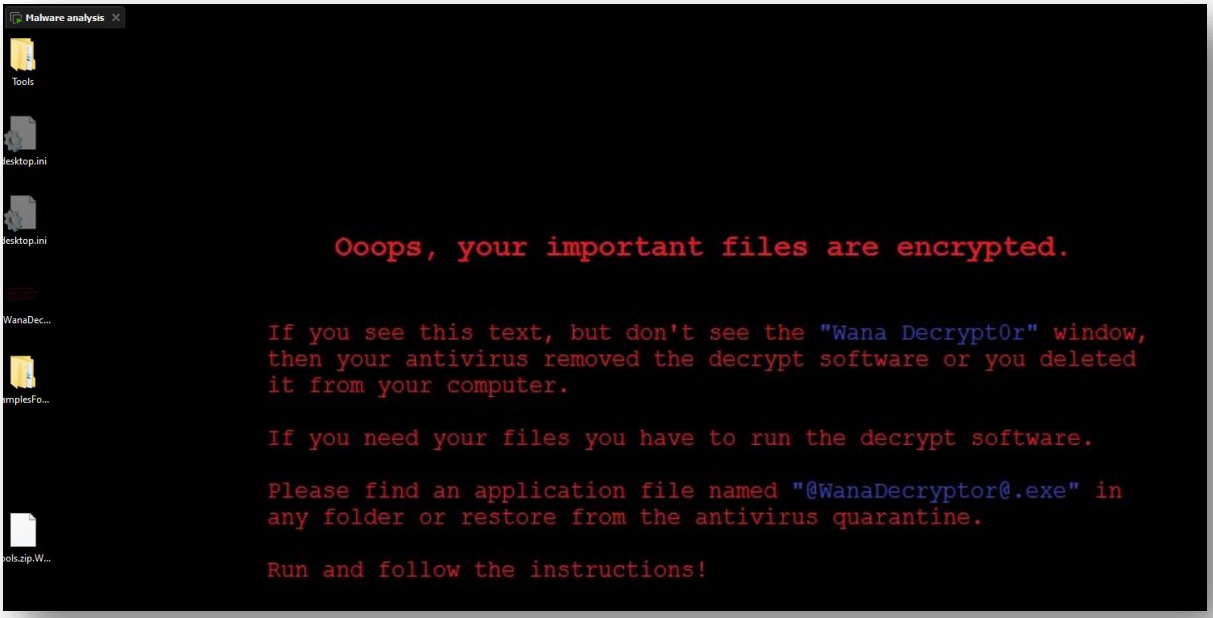

Figure 38 New desktop background.

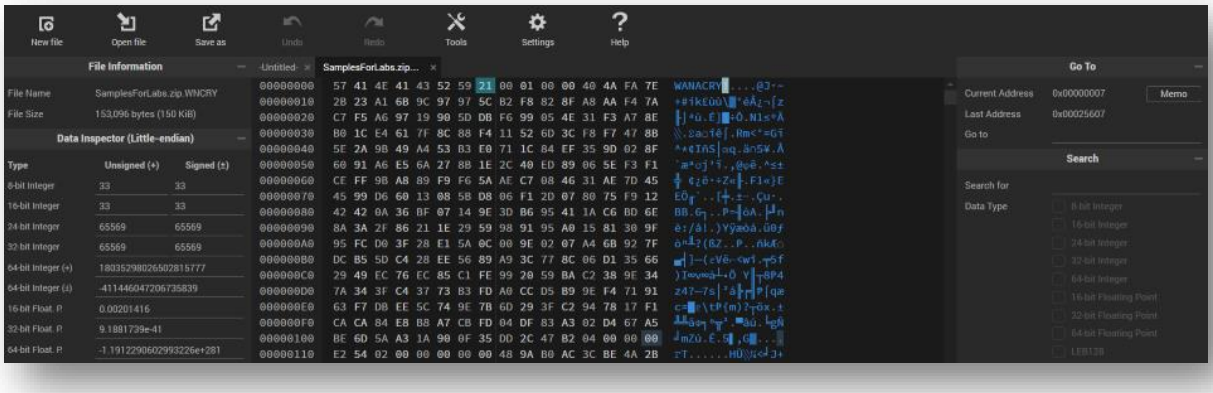

Figure 39 Start of every encrypted file.

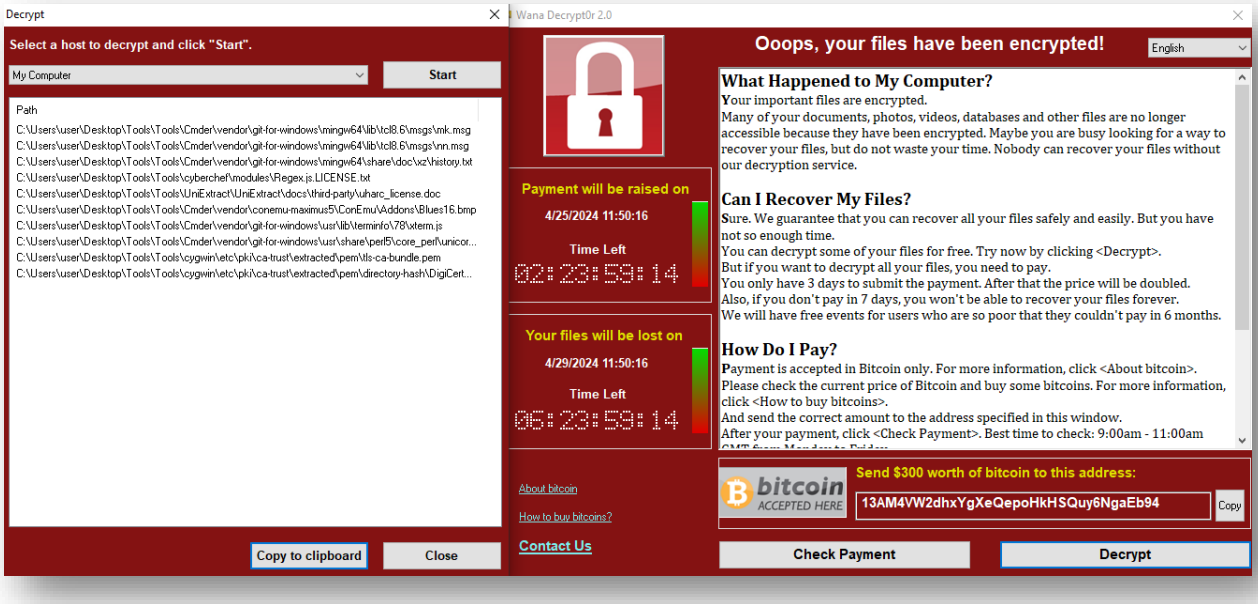

Figure 40 View of Decryptor and testing decrypt function.

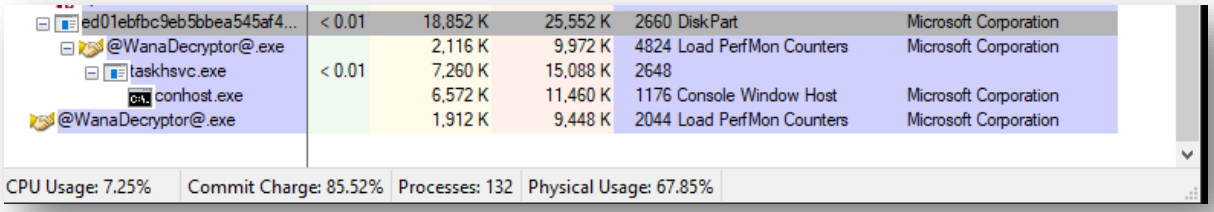

Figure 41 Initial processes which ran after executing the malware.

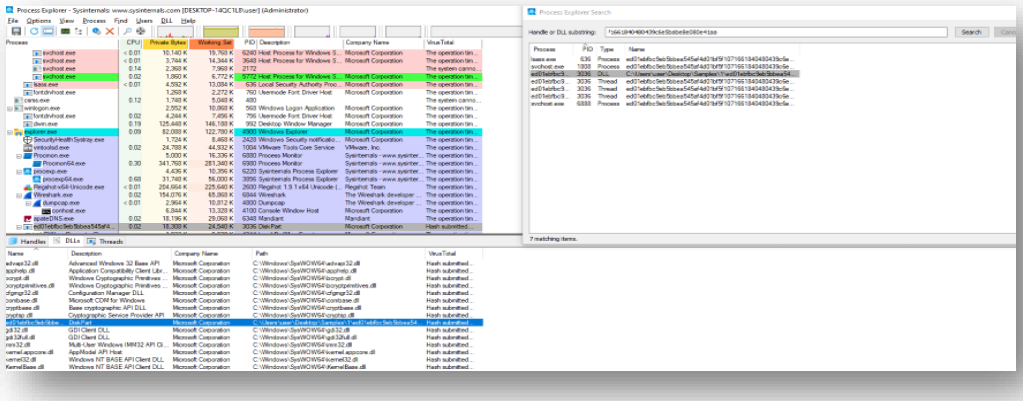

Figure 42 Search for the process, listing DLL's and Threads which the process uses.

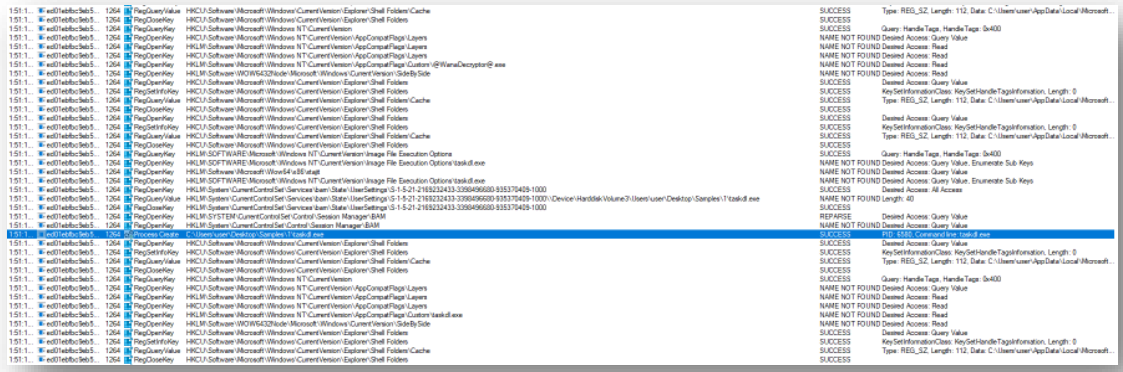

Figure 43 ProcMon process 1264.

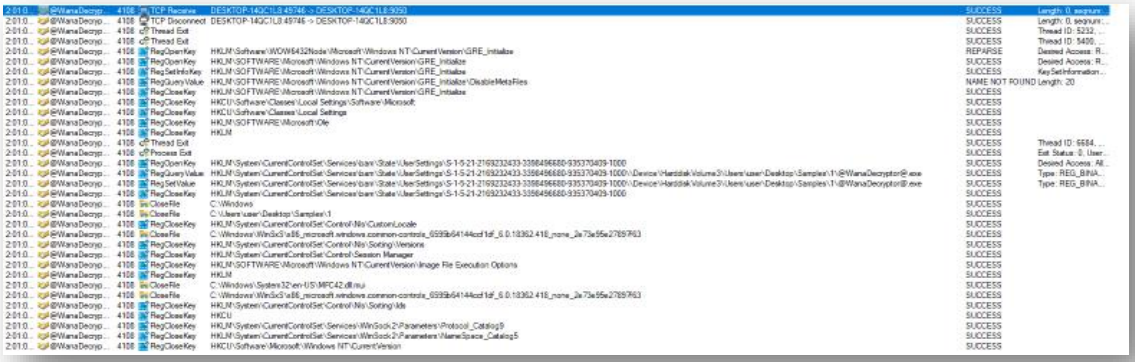

Figure 44 ProcMon process 4108.

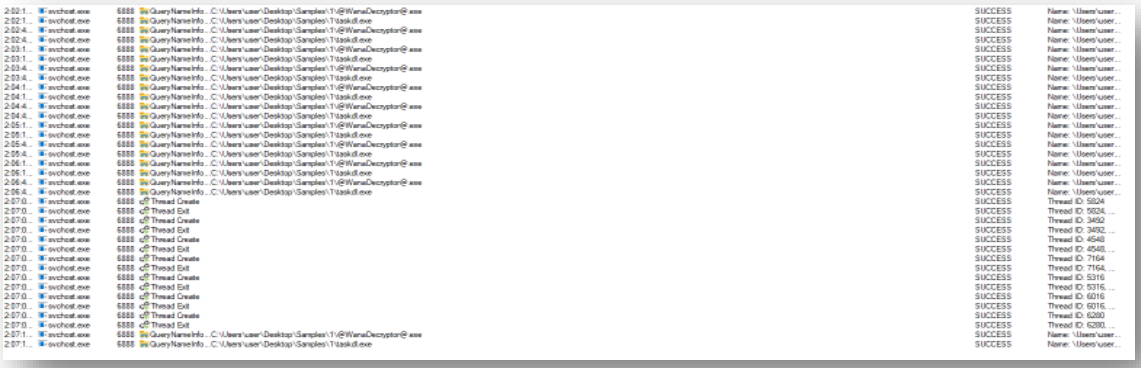

Figure 45 ProcMon process 6888.

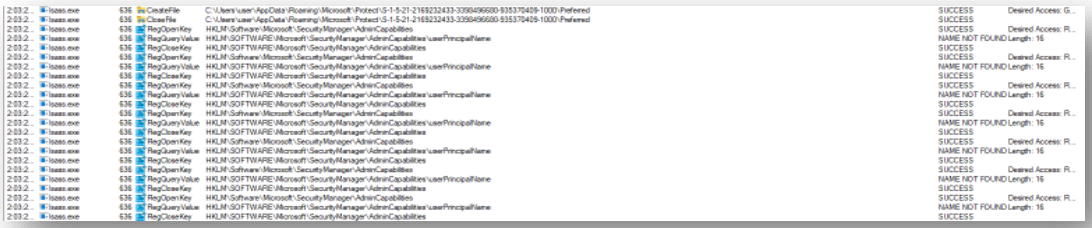

Figure 46 ProcMon process 636 part1.

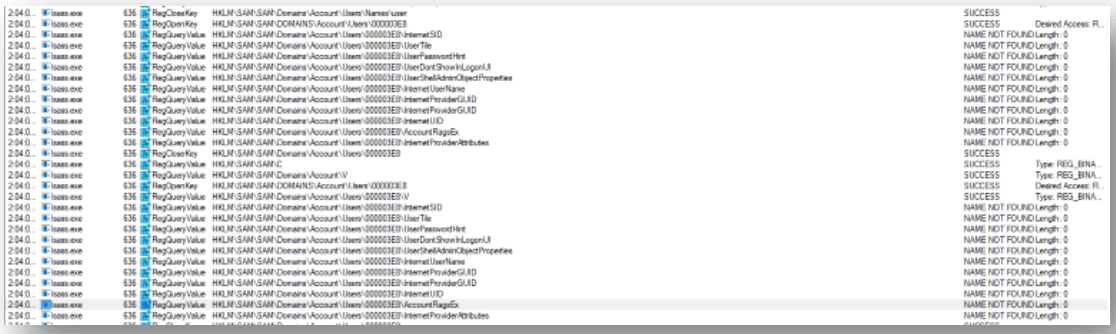

Figure 47 ProcMon process 636 part2.

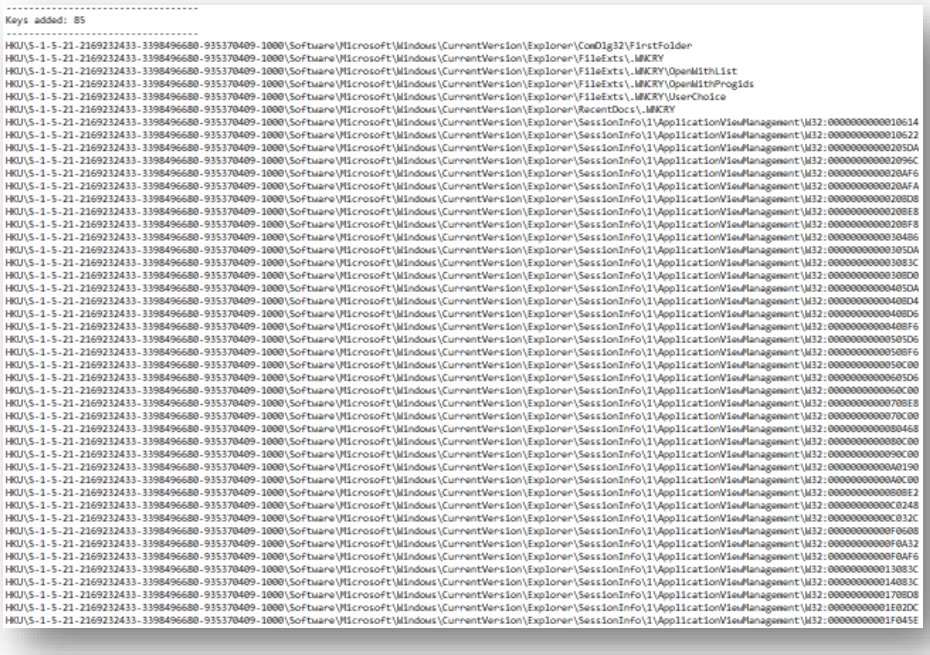

Figure 48 Regshot keys added part1.

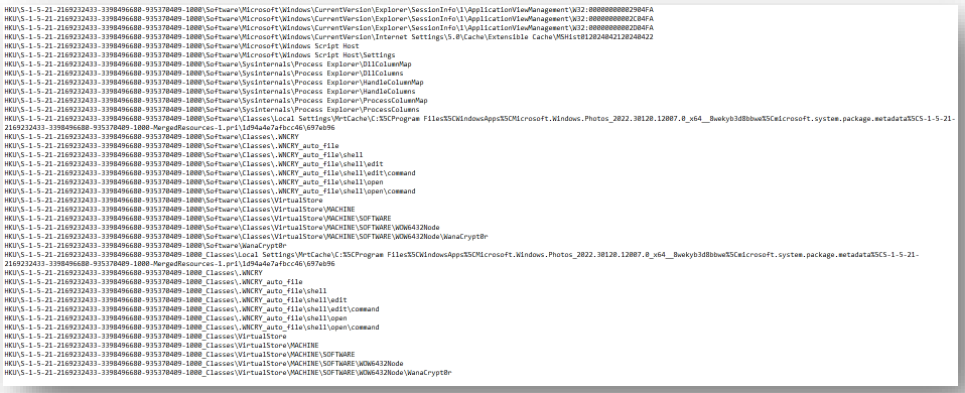

Figure 49 Regshot keys added part2.

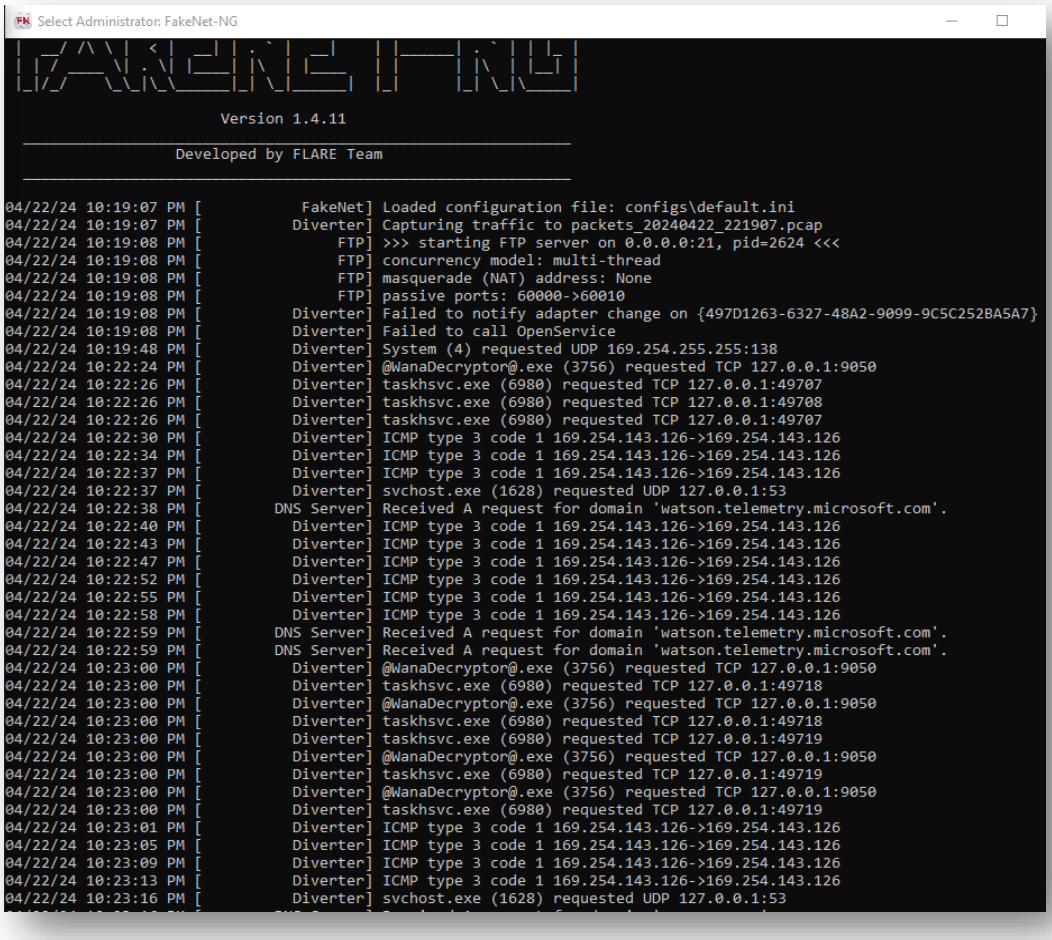

Figure 50 Fakenet results part1.

```
04/22/24 10:23:18 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:20 PM [        DNS Server] Received A request for domain 'watson.telemetry.microsoft.com'.
04/22/24 10:23:22 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:26 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:29 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:33 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:36 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:39 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:41 PM [        DNS Server] Received A request for domain 'watson.telemetry.microsoft.com'.
04/22/24 10:23:43 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:48 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:49 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:55 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:23:59 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:00 PM [          Diverter] @WanaDecryptor@.exe (3756) requested TCP 127.0.0.1:9050
04/22/24 10:24:00 PM [          Diverter] taskhsvc.exe (6980) requested TCP 127.0.0.1:49719
04/22/24 10:24:00 PM [          Diverter] taskhsvc.exe (6980) requested TCP 127.0.0.1:49728
04/22/24 10:24:00 PM [          Diverter] taskhsvc.exe (6980) requested TCP 127.0.0.1:49719
04/22/24 10:24:00 PM [          Diverter] @WanaDecryptor@.exe (3756) requested TCP 127.0.0.1:9050
04/22/24 10:24:00 PM [          Diverter] taskhsvc.exe (6980) requested TCP 127.0.0.1:49728
04/22/24 10:24:00 PM [          Diverter] @WanaDecryptor@.exe (3756) requested TCP 127.0.0.1:9050
04/22/24 10:24:00 PM [          Diverter] taskhsvc.exe (6980) requested TCP 127.0.0.1:49728
04/22/24 10:24:02 PM [          Diverter] svchost.exe (1628) requested UDP 127.0.0.1:53
04/22/24 10:24:02 PM [        DNS Server] Received A request for domain 'watson.telemetry.microsoft.com'.
04/22/24 10:24:05 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:08 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:12 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:15 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:20 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:24 PM [        DNS Server] Received A request for domain 'watson.telemetry.microsoft.com'.
04/22/24 10:24:24 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:27 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:33 PM [          Diverter] ICMP type 3 code 1 169.254.143.126->169.254.143.126
04/22/24 10:24:40 PM [           FakeNet] Stopping...
```

Figure 51 Fakenet results part2.

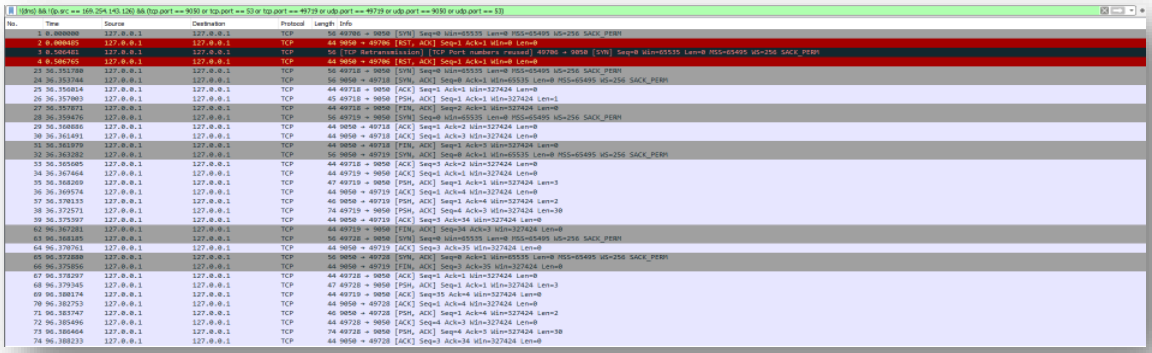

Figure 52 Wireshark packet capture.

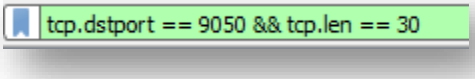

Figure 53 Applied filter to find onion addresses.

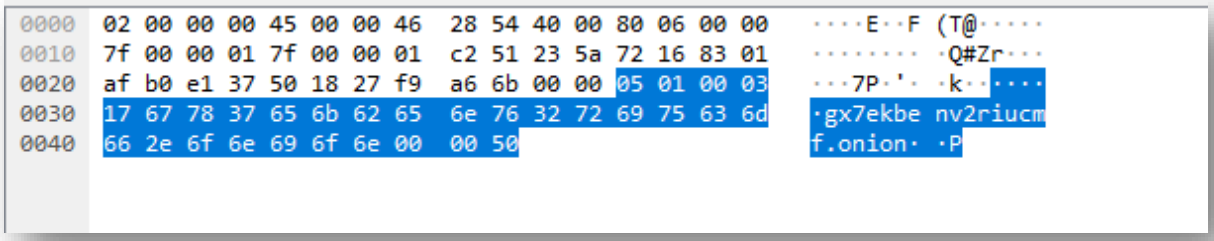

Figure 54 First onion address .

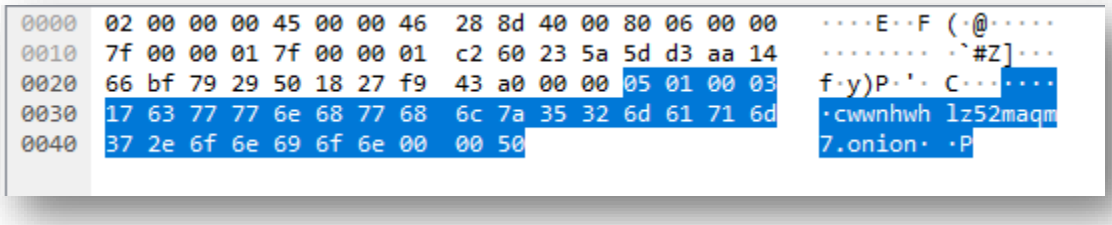

Figure 55 Second onion address.

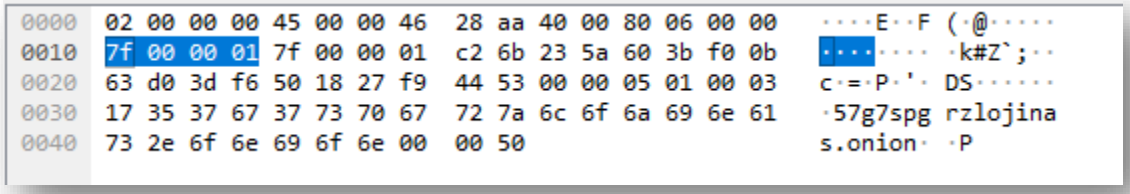

Figure 56 Third onion address.

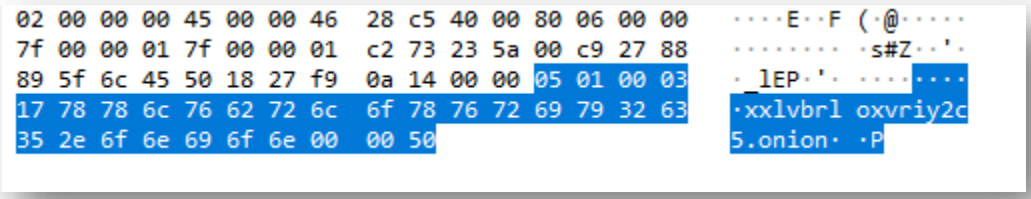

Figure 57 Forth onion address.

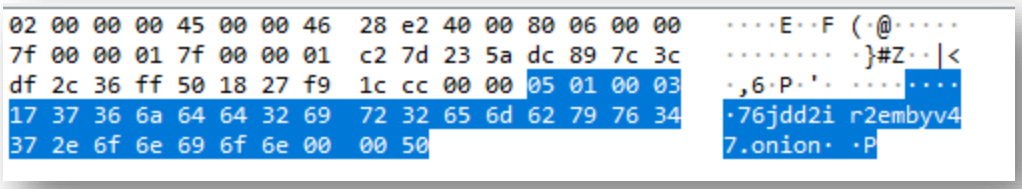

Figure 58 Fifth onion address.

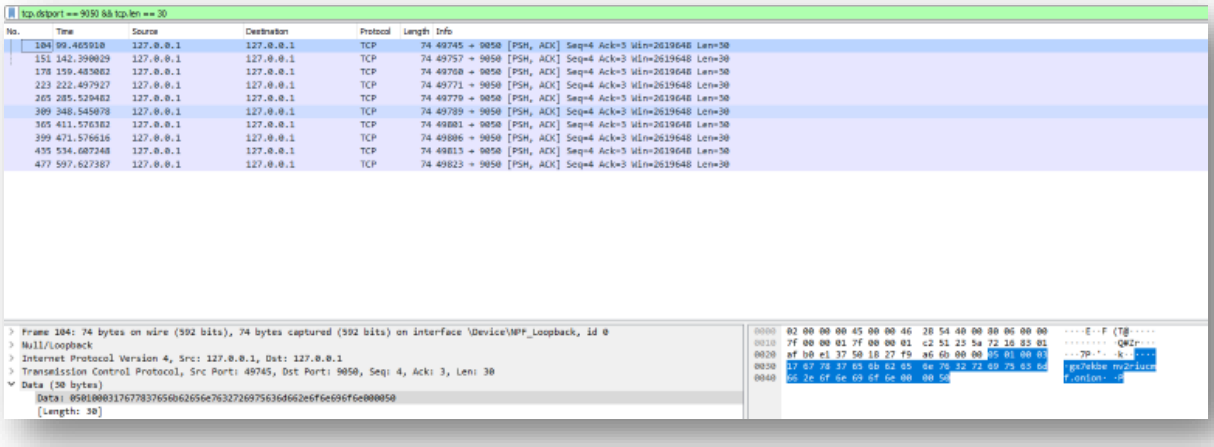

Figure 59 All results from filter.

APPENDIX B – LARGE IMAGES

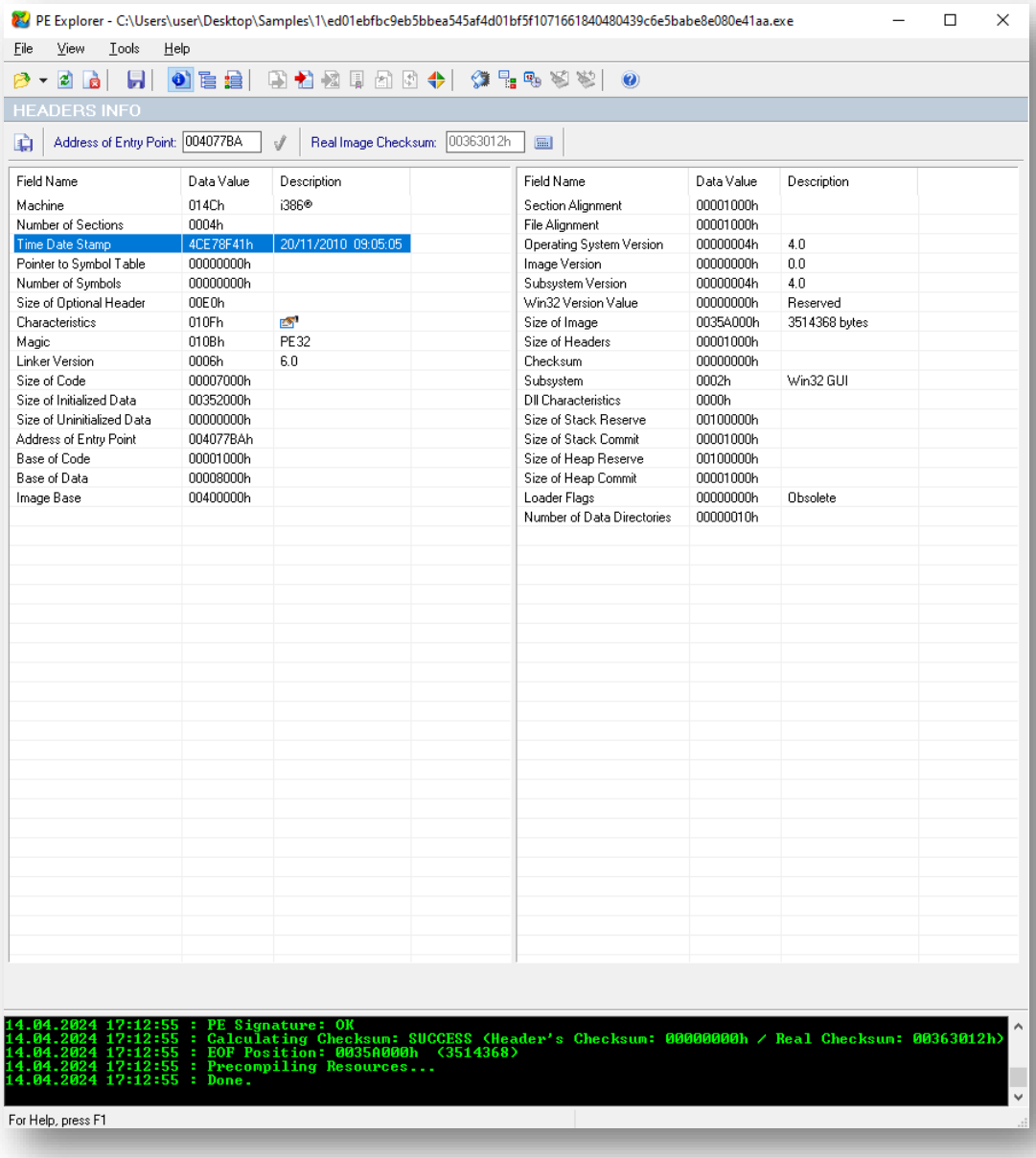

Figure 60 PE Explorer header information.

```
C:\Users\user\Desktop\Tools\Tools>upx.exe C:\Users\user\Desktop\Samples\1\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e08
0e41aa.exe -o file
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91w       Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: C:\Users\user\Desktop\Samples\1\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe: NotCompressibleException

Packed 1 file: 0 ok, 1 error.

C:\Users\user\Desktop\Tools\Tools>upx.exe -t C:\Users\user\Desktop\Samples\1\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8
e080e41aa.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91w       Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

upx: C:\Users\user\Desktop\Samples\1\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe: NotPackedException: not p
acked by UPX

Tested 0 files.

C:\Users\user\Desktop\Tools\Tools>
```

Figure 61 UPX.exe results on malware.

Decompile: WinMain - (ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe)

```
1
2  int WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
3
4  {
5    bool bVar1;
6    int *argv;
7    char ***local_EAX_89;
8    undefined3 extraout_var;
9    DWORD DVar2;
10   char *pcVar3;
11   short *psVar4;
12   code *pcVar5;
13   int iVar6;
14   undefined4 *puVar7;
15   undefined local_6e8 [1240];
16   char filename [520];
17   uint local_8;
18   char **_s_/i;
19
20   filename[0] = DAT_0040f910;
21   puVar7 = (undefined4 *)(filename + 1);
22   for (iVar6 = 0x81; iVar6 != 0; iVar6 = iVar6 + -1) {
23     *puVar7 = 0;
```

Figure 62 WinMain function part 1.

```
*(undefined2 *)puVar7 = 0;
*(undefined *)((int)puVar7 + 2) = 0;
GetModuleFileNameA((HMODULE)0x0,filename,520);
randomString_based_on_computer_name((int)&lpMultiByteStr_0040f8ac);
                    /* if less than 1 arg quit */
argv = (int *)__p___argc();
if (*argv == 2) {
  _s_/i = &_Str2_0040f538;
  local_EAX_89 = (char ***)__p___argv();
  iVar6 = strcmp((*local_EAX_89)[1],(char *)_s_/i);
  if ((iVar6 == 0) &&
     (bVar1 = create_and_cwd_random_hidden_dir((wchar_t *)0x0), CONCAT31(extraout_var,bVar1) != 0)
     ) {
    CopyFileA(filename,s_tasksche.exe_0040f4d8,0);
    DVar2 = GetFileAttributesA(s_tasksche.exe_0040f4d8);
    if ((DVar2 != 0xffffffff) && (iVar6 = create_or_start_tasksche_service(), iVar6 != 0)) {
      return 0;
    }
```

Figure 63 WinMain function part 2.

```
software_str = (undefined4 *)u_Software\_0040e04c;
puVar3 = software_str_buf;
for (iVar2 = 5; iVar2 != 0; iVar2 = iVar2 + -1) {
  *puVar3 = *software_str;
  software_str = software_str + 1;
  puVar3 = puVar3 + 1;
}
reg_vslue = '\0';
REGISTRY_WANA_HANDLE = (HKEY)0x0;
software_str = local_c4;
for (iVar2 = 0x2d; iVar2 != 0; iVar2 = iVar2 + -1) {
  *software_str = 0;
  software_str = software_str + 1;
}
software_str = &local_2df;
for (iVar2 = 0x81; iVar2 != 0; iVar2 = iVar2 + -1) {
  *software_str = 0;
  software_str = software_str + 1;
}
*(undefined2 *)software_str = 0;
*(undefined *)((int)software_str + 2) = 0;
                    /* Software\WanaCrypt0r */
wcscat((wchar_t *)software_str_buf,u_WanaCrypt0r_0040e034);
```

Figure 64 set_or_query_reg_cwd function part 1.

```
i = 0;
do {
  if (i == 0) {
                    /* HKEY_LOCAL_MACHINE */
    hKey = (HKEY)0x80000002;
  }
  else {
                    /* HKEY_CURRENT_USER */
    hKey = (HKEY)0x80000001;
  }
  RegCreateKeyW(hKey,(LPCWSTR)software_str_buf,&REGISTRY_WANA_HANDLE);
  if (REGISTRY_WANA_HANDLE != (HKEY)0x0) {
    if (set_registry == 0) {
      local_10 = 0x207;
      LVar1 = RegQueryValueExA(REGISTRY_WANA_HANDLE,&DAT_0040e030,(LPDWORD)0x0,(LPDWORD)0x0,
                               &reg_vslue,&local_10);
      bVar4 = LVar1 == 0;
      if (bVar4) {
        SetCurrentDirectoryA((LPCSTR)&reg_vslue);
      }
    }
    else {
      GetCurrentDirectoryA(0x207,(LPSTR)&reg_vslue);
      current_directory_length = strlen((char *)&reg_vslue);
      LVar1 = RegSetValueExA(REGISTRY_WANA_HANDLE,&DAT_0040e030,0,1,&reg_vslue,
                             current_directory_length + 1);
      bVar4 = LVar1 == 0;
    }
    RegCloseKey(REGISTRY_WANA_HANDLE);
    if (bVar4) {
      return 1;
    }
  }
```

Figure 65 set_or_query_reg_cwd function part 2.

APPENDIX C - EXTRA TABLES

Hashes for: ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe	
MD5	84c82835a5d21bbcf75a61706d8ab549
SHA1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA-256	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
SHA-384	d7e90fcb0830b2be200b6cc9d86484efe8df14fc1604c84179d21e283f710b4c248c0ac43b992b955fa394bd20a8083c
SHA-512	90723a50c20ba3643d625595fd6be8dcf88d70ff7f4b4719a88f055d5b3149a423
CRC32	4022fcaa

Figure 66 Tables of hashes for software.

Popular threat label	ransomware.wannacry/wannacryptor	Threat categories	ransomware, trojan	Family labels	wannacry, wannacryptor, wanr

Security vendors' analysis			Do you want to automate checks?
AhnLab-V3	Trojan/Win32.WannaCryptor.R200571	Alibaba	Ransom:Win32/WannaCry.ali1020010
AliCloud	RansomWare	ALYac	Trojan.Ransom.WannaCryptor
Antiy-AVL	Trojan[Ransom]/Win32.Scatter	Arcabit	Trojan.Ransom.WannaCryptor.A
Avast	Win32:WanaCry-A [Trj]	AVG	Win32:WanaCry-A [Trj]
Avira (no cloud)	TR/Ransom.JB	Baidu	Win32.Trojan.WannaCry.c
BitDefender	Trojan.Ransom.WannaCryptor.A	BitDefenderTheta	Gen:NN.ZexaF.36802.wt0@aGEmS3di
Bkav Pro	W32.WanaCryptBTTc.Worm	ClamAV	Win.Ransomware.Wannacryptor-99401…
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	Cylance	Unsafe
Cynet	Malicious (score: 100)	DeepInstinct	MALICIOUS
DrWeb	Trojan.Encoder.11432	Elastic	Malicious (high Confidence)
Emsisoft	Trojan.Ransom.WannaCryptor.A (B)	eScan	Trojan.Ransom.WannaCryptor.A
ESET-NOD32	Win32/Filecoder.WannaCryptor.D	Fortinet	W32/WannaCryptor.6F87!tr.ransom
GData	Win32.Trojan-Ransom.WannaCry.A	Gridinsoft (no cloud)	Ransom.Win32.Filecoder.dd
Ikarus	Trojan-Ransom.WannaCry	Jiangmin	Trojan.Wanna.eo
K7AntiVirus	Trojan (0050d7171)	K7GW	Trojan (0050d7171)
Kaspersky	Trojan-Ransom.Win32.Wanna.zbu	Kingsoft	Win32.Troj.Undef.a
Malwarebytes	Generic.Malware.AI.DDS	MAX	Malware (ai Score=100)
McAfee	Ransom-O.g	Microsoft	Ransom:Win32/WannaCrypt
NANO-Antivirus	Trojan.Win32.Ransom.eoptnj	Panda	Trj/RansomCrypt.K
QuickHeal	Ransom.WannaCrypt.A4	Rising	Ransom.WanaCrypt!1.AAEB (CLASSIC)
Sangfor Engine Zero	Ransom.Win32.Save.WannaCry	SecureAge	Malicious
SentinelOne (Static ML)	Static AI - Malicious PE	Skyhigh (SWG)	BehavesLike.Win32.RansomWannaCry.wc
Sophos	Troj/Ransom-EMG	Symantec	Ransom.Wannacry
TACHYON	Ransom/W32.WannaCry.Zen	TEHTRIS	Generic.Malware
Tencent	Trojan-Ransom.Win32.WannaCry.kd	Trapmine	Malicious.high.ml.score
Trellix (FireEye)	Generic.mg.84c82835a5d21bbc	TrendMicro	Ransom_WANA.A
TrendMicro-HouseCall	Ransom_WANA.A	Varist	W32/Trojan.ZTSA-8671
VBA32	TrojanRansom.WannaCrypt	VIPRE	Trojan.Ransom.WannaCryptor.A
VirIT	Trojan.Win32.WannaCry.B	ViRobot	Trojan.Win32.S.WannaCry.3514368.N
WithSecure	Trojan.TR/Ransom.JB	Xcitium	Malware@#4gwtqo9z2tkf
Yandex	Trojan.Igent.bUj9pX.12	Zillya	Trojan.WannaCry.Win32.2
ZoneAlarm by Check Point	Trojan-Ransom.Win32.Wanna.zbu	Zoner	Trojan.Win32.55605
Acronis (Static ML)	Undetected	CMC	Undetected
Google	Undetected	MaxSecure	Undetected
Palo Alto Networks	Undetected	SUPERAntiSpyware	Undetected

Figure 67 VirusTotal.com results - Full list.

APPENDIX D – IDENTIFIED IMPORTS

RVA	Name
0040DBA...	KERNEL32.dll
0040DBC...	USER32.dll
0040DC84h	ADVAPI32.dll
0040DE88h	MSVCRT.dll

RVA	Hint	Name
0040802Ch	0061h	GetFileAttributesW
00408030h	0064h	GetFileSizeEx
00408034h	0053h	CreateFileA
00408038h	0023h	InitializeCriticalSection
0040803Ch	0081h	DeleteCriticalSection
00408040h	00B5h	ReadFile
00408044h	0063h	GetFileSize
00408048h	00A4h	WriteFile
0040804Ch	0051h	LeaveCriticalSection
00408050h	0098h	EnterCriticalSection
00408054h	001Ah	SetFileAttributesW
00408058h	000Bh	SetCurrentDirectoryW
0040805Ch	004Eh	CreateDirectoryW
00408060h	00D6h	GetTempPathW
00408064h	00F4h	GetWindowsDirectoryW
00408068h	005Eh	GetFileAttributesA
0040806Ch	0055h	SizeofResource
00408070h	0065h	LockResource
00408074h	0057h	LoadResource
00408078h	0075h	MultiByteToWideChar
0040807Ch	0056h	Sleep
00408080h	0084h	OpenMutexA
00408084h	0069h	GetFullPathNameA
00408088h	0043h	CopyFileA
0040808Ch	007Dh	GetModuleFileNameA
00408090h	0081h	VirtualAlloc
00408094h	0083h	VirtualFree
00408098h	00F8h	FreeLibrary
0040809Ch	0010h	HeapAlloc
004080A0h	00A3h	GetProcessHeap
004080A4h	007Fh	GetModuleHandleA
004080A8h	0028h	SetLastError
004080ACh	0086h	VirtualProtect
004080B0h	0033h	IsBadReadPtr
004080B4h	0016h	HeapFree
004080B8h	005Bh	SystemTimeToFileTime
004080BCh	005Ah	LocalFileTimeToFileTime
004080C0h	004Bh	CreateDirectoryA
004080C4h	00B7h	GetStartupInfoA
004080C8h	001Bh	SetFilePointer
004080CCh	001Fh	SetFileTime
004080D0h	0017h	GetComputerNameW
004080D4h	0040h	GetCurrentDirectoryA
004080D8h	000Ah	SetCurrentDirectoryA
004080DCh	00F8h	GlobalAlloc
004080E0h	0052h	LoadLibraryA
004080E4h	00A0h	GetProcAddress
004080E8h	00FFh	GlobalFree
004080ECh	0066h	CreateProcessA
004080F0h	0034h	CloseHandle
004080F4h	0090h	WaitForSingleObject
004080F8h	005Eh	TerminateProcess
004080FCh	005Ah	GetExitCodeProcess
00408100h	00E3h	FindResourceA

Figure 68 KERNAL32.dll imports.

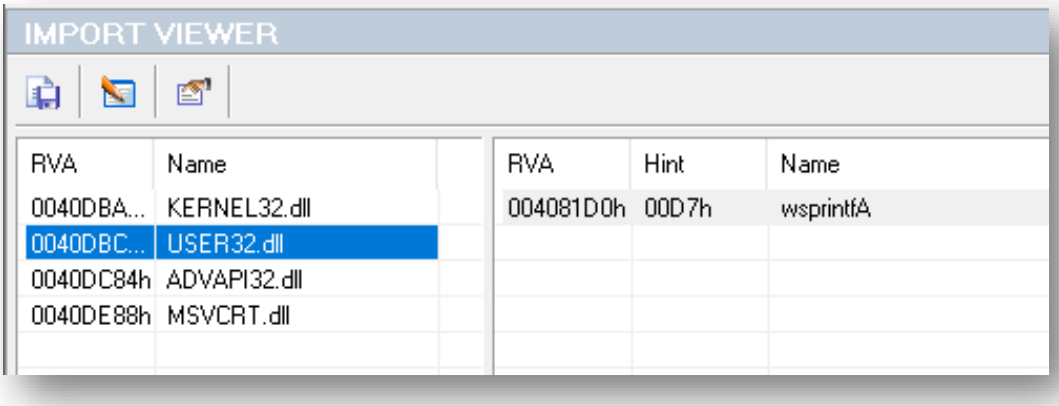

Figure 69 USER32.dll imports.

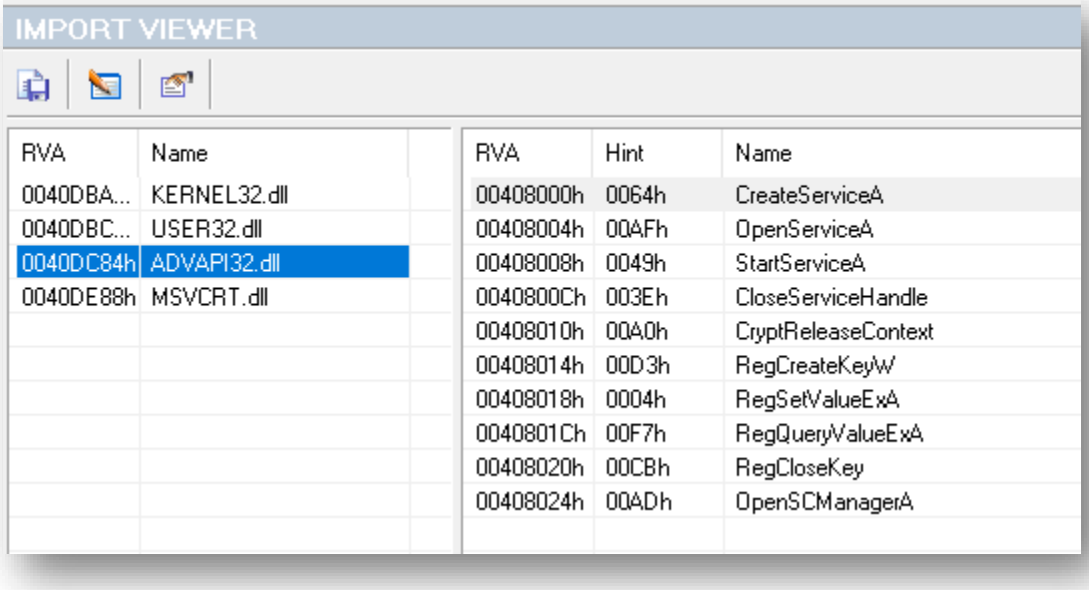

Figure 70 ADVAPI32.dll imports.

Figure 71 MSVCRT.dll imports.

APPENDIX E – MEMORY ANALYSIS

```
┌──(kali㉿kali)-[~/volatility3]
└─$ sudo python3 vol.py -f /home/kali/Desktop/infectedSNAP.vmem windows.pslist
Volatility 3 Framework 2.7.0
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        File output

4       0       System  0×998f4de6d040  123     -       N/A     False   2024-03-12 17:25:15.000000      N/A     Disabled
88      4       Registry        0×998f4dec6080  4       -       N/A     False   2024-03-12 17:25:11.000000      N/A     Disabled
288     4       smss.exe        0×998f4ecd3480  2       -       N/A     False   2024-03-12 17:25:15.000000      N/A     Disabled
392     384     csrss.exe       0×998f4edd23c0  10      -       0       False   2024-03-12 17:25:16.000000      N/A     Disabled
472     384     wininit.exe     0×998f51a22080  1       -       0       False   2024-03-12 17:25:16.000000      N/A     Disabled
480     464     csrss.exe       0×998f4edca0c0  11      -       1       False   2024-03-12 17:25:16.000000      N/A     Disabled
568     464     winlogon.exe    0×998f51a7d200  3       -       1       False   2024-03-12 17:25:16.000000      N/A     Disabled
608     472     services.exe    0×998f51a8c080  5       -       0       False   2024-03-12 17:25:16.000000      N/A     Disabled
636     472     lsass.exe       0×998f51b47080  7       -       0       False   2024-03-12 17:25:16.000000      N/A     Disabled
740     608     svchost.exe     0×998f52213080  1       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
756     568     fontdrvhost.ex  0×998f52219080  5       -       1       False   2024-03-12 17:25:17.000000      N/A     Disabled
760     472     fontdrvhost.ex  0×998f5221a080  5       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
824     608     svchost.exe     0×998f5222b080  12      -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
880     608     svchost.exe     0×998f52287240  9       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
924     608     svchost.exe     0×998f522af080  5       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
992     568     dwm.exe 0×998f522d4080  14      -       1       False   2024-03-12 17:25:17.000000      N/A     Disabled
772     608     svchost.exe     0×998f523850c0  4       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1028    608     svchost.exe     0×998f523ba080  2       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1044    608     svchost.exe     0×998f523bb080  3       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1120    608     svchost.exe     0×998f524020c0  1       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1128    608     svchost.exe     0×998f52405080  1       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1160    608     svchost.exe     0×998f524042c0  7       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1212    608     svchost.exe     0×998f52407080  6       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1268    608     svchost.exe     0×998f52495080  1       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1296    608     svchost.exe     0×998f52498080  5       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1408    608     svchost.exe     0×998f52509080  3       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1432    608     svchost.exe     0×998f5250a080  4       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1488    608     svchost.exe     0×998f52543080  5       -       0       False   2024-03-12 17:25:17.000000      N/A     Disabled
1592    608     svchost.exe     0×998f4de68080  6       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1628    608     svchost.exe     0×998f4df11080  9       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1776    608     svchost.exe     0×998f4dfb7080  3       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1784    608     svchost.exe     0×998f4dfb5080  7       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1808    608     svchost.exe     0×998f4dfb3080  3       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1852    608     svchost.exe     0×998f4df80080  5       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1904    608     svchost.exe     0×998f4df7a080  2       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1920    4       MemCompression  0×998f4df77040  22      -       N/A     False   2024-03-12 17:25:18.000000      N/A     Disabled
2004    608     svchost.exe     0×998f526c3080  2       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2012    608     svchost.exe     0×998f526c4080  7       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
1468    608     svchost.exe     0×998f526e74c0  2       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2056    608     svchost.exe     0×998f5274a080  9       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
```

Figure 72 Memory analysis of processes part1.

```
2092    608     svchost.exe     0×998f5277a080  3       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2152    608     svchost.exe     0×998f52795080  7       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2224    608     svchost.exe     0×998f527f7080  2       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2316    608     svchost.exe     0×998f52832080  11      -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2468    608     svchost.exe     0×998f528c58c0  2       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2480    608     spoolsv.exe     0×998f528c98c0  7       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2576    608     svchost.exe     0×998f528ee080  5       -       0       False   2024-03-12 17:25:18.000000      N/A     Disabled
2640    608     svchost.exe     0×998f5294f080  16      -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2720    608     svchost.exe     0×998f51010080  4       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2728    608     svchost.exe     0×998f51011080  4       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2736    608     svchost.exe     0×998f510130c0  3       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2868    608     svchost.exe     0×998f52a3d080  5       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2892    608     svchost.exe     0×998f52a54080  14      -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2976    608     svchost.exe     0×998f52aae080  6       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
2996    608     svchost.exe     0×998f52abd400  3       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3024    608     VGAuthService.  0×998f52acb0c0  2       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3040    608     vm3dservice.ex  0×998f52ad2080  2       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3048    608     vmtoolsd.exe    0×998f52ae4080  13      -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3060    608     svchost.exe     0×998f52ae5080  3       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3148    3040    vm3dservice.ex  0×998f52bc40c0  2       -       1       False   2024-03-12 17:25:19.000000      N/A     Disabled
3260    608     svchost.exe     0×998f52bf4080  2       -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3568    608     dllhost.exe     0×998f52d5b080  11      -       0       False   2024-03-12 17:25:19.000000      N/A     Disabled
3676    608     msdtc.exe       0×998f52d74080  9       -       0       False   2024-03-12 17:25:20.000000      N/A     Disabled
4016    824     WmiPrvSE.exe    0×998f52e28080  10      -       0       False   2024-03-12 17:25:20.000000      N/A     Disabled
3476    608     svchost.exe     0×998f52fa7080  5       -       0       False   2024-03-12 17:25:28.000000      N/A     Disabled
2220    608     svchost.exe     0×998f52e5b080  0       -       0       False   2024-03-12 17:25:40.000000      2024-04-22 18:52:20.000000      Disabled
3524    608     svchost.exe     0×998f52bc7080  7       -       0       False   2024-03-12 17:25:40.000000      N/A     Disabled
4268    1432    sihost.exe      0×998f530f2080  6       -       1       False   2024-03-12 17:25:46.000000      N/A     Disabled
4284    608     svchost.exe     0×998f530f8080  7       -       1       False   2024-03-12 17:25:46.000000      N/A     Disabled
4328    608     svchost.exe     0×998f53120080  2       -       1       False   2024-03-12 17:25:46.000000      N/A     Disabled
4372    1212    MicrosoftEdgeU  0×998f53130080  3       -       0       True    2024-03-12 17:25:46.000000      N/A     Disabled
4388    1212    GoogleUpdate.e  0×998f53135080  4       -       0       True    2024-03-12 17:25:46.000000      N/A     Disabled
4468    608     svchost.exe     0×998f5318f080  3       -       0       False   2024-03-12 17:25:46.000000      N/A     Disabled
4572    608     svchost.exe     0×998f5312a080  3       -       0       False   2024-03-12 17:25:46.000000      N/A     Disabled
4612    1212    taskhostw.exe   0×998f532020c0  7       -       1       False   2024-03-12 17:25:46.000000      N/A     Disabled
4620    4572    ctfmon.exe      0×998f53208080  8       -       1       False   2024-03-12 17:25:46.000000      N/A     Disabled
4732    608     svchost.exe     0×998f53246080  5       -       0       False   2024-03-12 17:25:46.000000      N/A     Disabled
4856    568     userinit.exe    0×998f532cb080  0       -       1       False   2024-03-12 17:25:46.000000      2024-03-12 17:26:10.000000      Disabled
4900    4856    explorer.exe    0×998f532fa080  60      -       1       False   2024-03-12 17:25:46.000000      N/A     Disabled
5036    608     svchost.exe     0×998f53386080  6       -       1       False   2024-03-12 17:25:47.000000      N/A     Disabled
4512    608     SearchIndexer.  0×998f53439080  15      -       0       False   2024-03-12 17:25:47.000000      N/A     Disabled
5168    824     StartMenuExper  0×998f533912c0  6       -       1       False   2024-03-12 17:25:47.000000      N/A     Disabled
5296    824     RuntimeBroker.  0×998f5366b080  2       -       1       False   2024-03-12 17:25:48.000000      N/A     Disabled
5440    824     SearchUI.exe    0×998f53651080  33      -       1       False   2024-03-12 17:25:48.000000      N/A     Disabled
5052    824     RuntimeBroker.  0×998f53807080  12      -       1       False   2024-03-12 17:25:48.000000      N/A     Disabled
5916    824     RuntimeBroker.  0×998f527ef0c0  2       -       1       False   2024-03-12 17:25:49.000000      N/A     Disabled
2428    4900    SecurityHealth  0×998f53137080  1       -       1       False   2024-03-12 17:25:59.000000      N/A     Disabled
5792    608     SecurityHealth  0×998f5313d080  9       -       0       False   2024-03-12 17:25:59.000000      N/A     Disabled
```

Figure 73 Memory analysis of processes part2.

Figure 74 Memory analysis taskhsvc.exe attempt to connect to 199.254.238.52 on port 443.

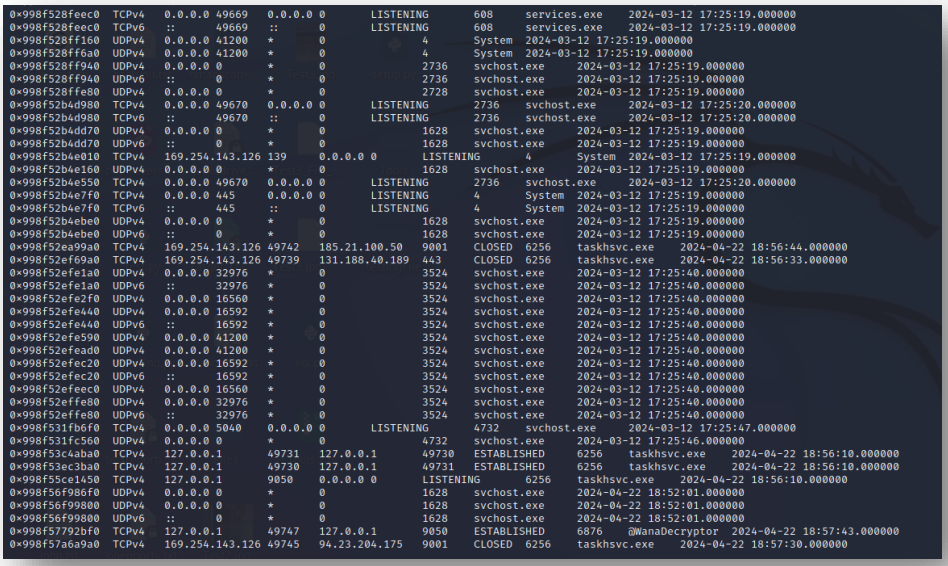

Figure 75 Memory analysis using windows.netscan.

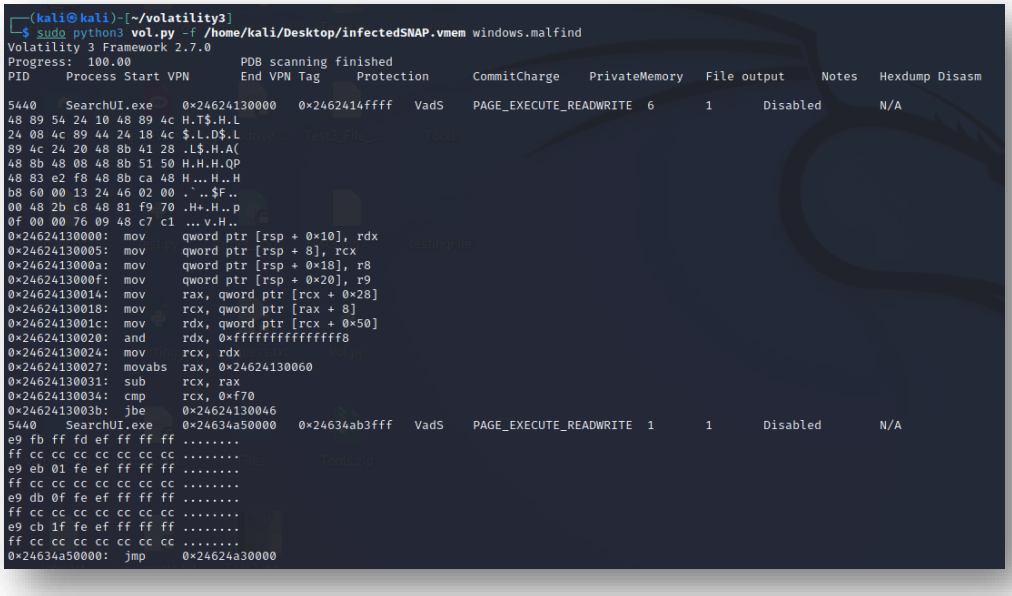

Figure 76 Memory analysis using windows.malfind.

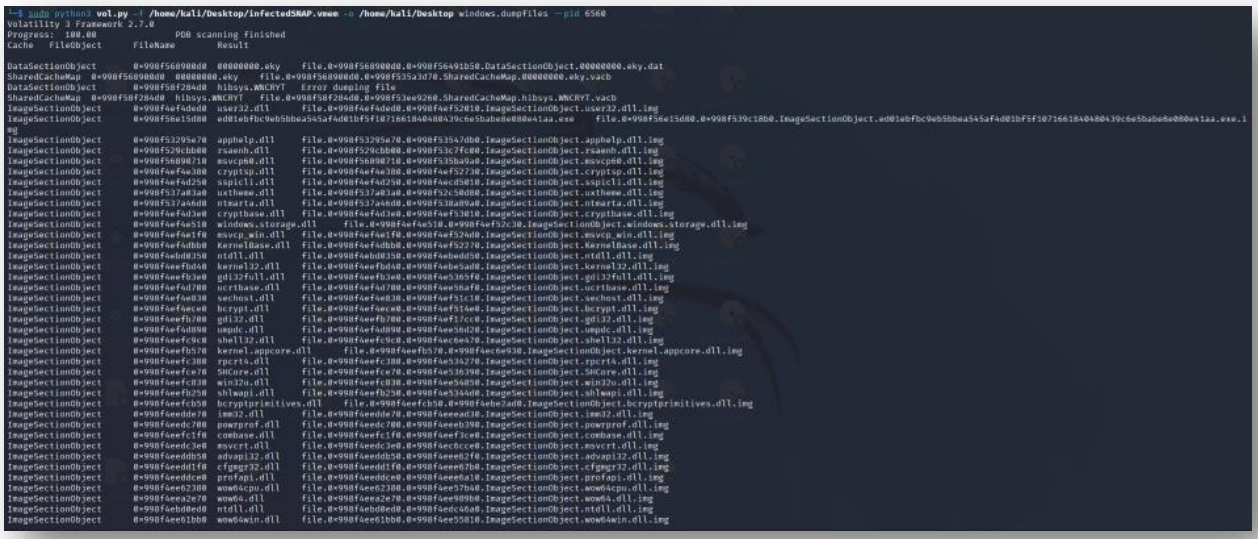

Figure 77 Memory analysis, dumping all files related to the main application with the PID of 6560.