Rustware Malware Analysis Report

Cyber Security Incident Response Team:

12/6/2023

Executive Summary

This malware analysis report provides a comprehensive examination of a recently discovered malicious software, aiming to offer insights into its functionality, potential impact, and recommended mitigation strategies. The analyzed malware, identified as Rustware, poses a significant threat to organizations and individuals due to its sophisticated nature and diverse attack vectors.

Case Details

Date	**12/6/2023**
Analyst	

Sample information

File name	**Rustware.exe**
File size	**283.20 mb**
File type	**Win32 EXE**
MD5	**f6c6b4070714b761242752dfdf8aa0f7**
SHA1	**ffa1c30d0d2856c09a19439e57125edc9dfce4e9**
SHA256	**ca9d23d4be4fc5c1bae0f57b681c462d31d7b817627a966981c329c848131 02e**
Packer / compiler info	**NA**
Compile time	**2023-12-06 03:31:05 UTC**

Analysis

What is malware analysis?
The study of the unique features, objectives, sources, and potential effects of harmful software and code, such as spyware, viruses, malvertising, and ransomware. It analyzes malware code to understand how it varies from other kinds.

There are several types of malware analysis. Static Malware Analysis looks for files that may harm your system without actively running the malware code, making it a safe tool for exposing malicious libraries or packaged files.Dynamic malware analysis uses a sandbox, which is a secure, isolated, virtual environment where you can run suspected dangerous code. Hybrid malware analysis combines both static and dynamic techniques.

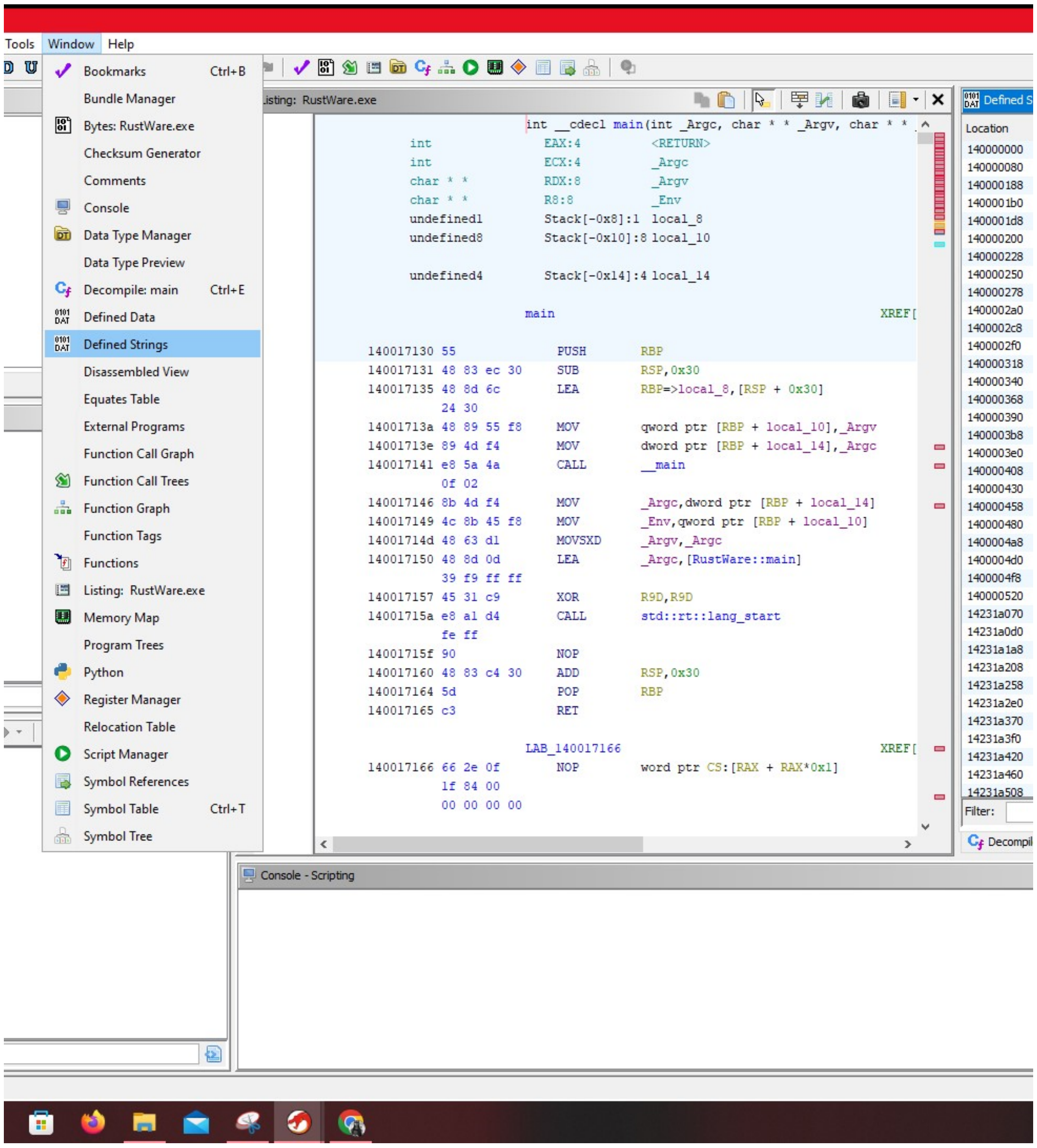
There are 4 stages with malware analysis ; static properties analysis refers to strings of code embedded inside the malware file, hashes, header details, and metadata. Interactive behavior analysis involves a security analyst interacting with malware running in a lab, making observations regarding its behavior. Fully automated analysis scans suspected malware files using automated tools, focusing on what the malware can do once inside your system. Manual code reversing breaks down the code used to build the malware to learn how it works and what it is capable of doing.

In this report we are going to go only over 3 out of the 4 stages of Malware Analysis. Which are Static Properties Analysis, Interactive Behavior Analysis, and Manual Code Reversing.

Static Properties Analysis

For our initial Static Properties Analysis I am going to use Ghidra a well known open source reversing engineer tool developed by the NSA.

Initially we would want to extract strings of code embedded inside the malware file that we have and Ghidra has a nice feature for that. So one of the very first thing you want to is load the binary on Ghidra and go to the windows options and click on defined strings like in Figure 1 below



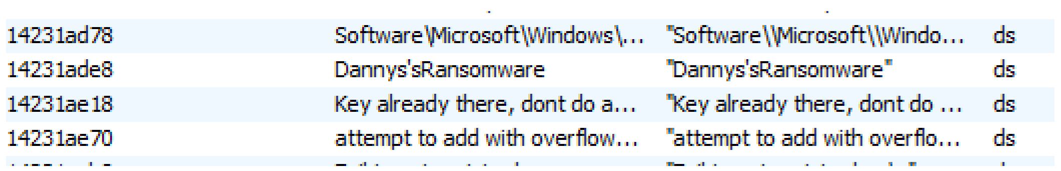
```
                       int __cdecl main(int _Argc, char * * _Argv, char * *
          int              EAX:4          <RETURN>
          int              ECX:4          _Argc
          char * *         RDX:8          _Argv
          char * *         R8:8           _Env
          undefined1       Stack[-0x8]:1  local_8
          undefined8       Stack[-0x10]:8 local_10

          undefined4       Stack[-0x14]:4 local_14

                       main                                       XREF[

     140017130 55            PUSH      RBP
     140017131 48 83 ec 30   SUB       RSP,0x30
     140017135 48 8d 6c      LEA       RBP=>local_8,[RSP + 0x30]
               24 30
     14001713a 48 89 55 f8   MOV       qword ptr [RBP + local_10],_Argv
     14001713e 89 4d f4      MOV       dword ptr [RBP + local_14],_Argc
     140017141 e8 5a 4a      CALL      __main
               0f 02
     140017146 8b 4d f4      MOV       _Argc,dword ptr [RBP + local_14]
     140017149 4c 8b 45 f8   MOV       _Env,qword ptr [RBP + local_10]
     14001714d 48 63 d1      MOVSXD    _Argv,_Argc
     140017150 48 8d 0d      LEA       _Argc,[RustWare::main]
               39 f9 ff ff
     140017157 45 31 c9      XOR       R9D,R9D
     14001715a e8 a1 d4      CALL      std::rt::lang_start
               fe ff
     14001715f 90            NOP
     140017160 48 83 c4 30   ADD       RSP,0x30
     140017164 5d            POP       RBP
     140017165 c3            RET

                       LAB_140017166                              XREF[
     140017166 66 2e 0f      NOP       word ptr CS:[RAX + RAX*0x1]
               1f 84 00
               00 00 00 00
```

```
140000000
140000080
140000188
1400001b0
1400001d8
140000200
140000228
140000250
140000278
1400002a0
1400002c8
1400002f0
140000318
140000340
140000368
140000390
1400003b8
1400003e0
140000408
140000430
140000458
140000480
1400004a8
1400004d0
1400004f8
140000520
14231a070
14231a0d0
14231a1a8
14231a208
14231a258
14231a2e0
14231a370
14231a3f0
14231a420
14231a460
14231a508
```

Figure 1

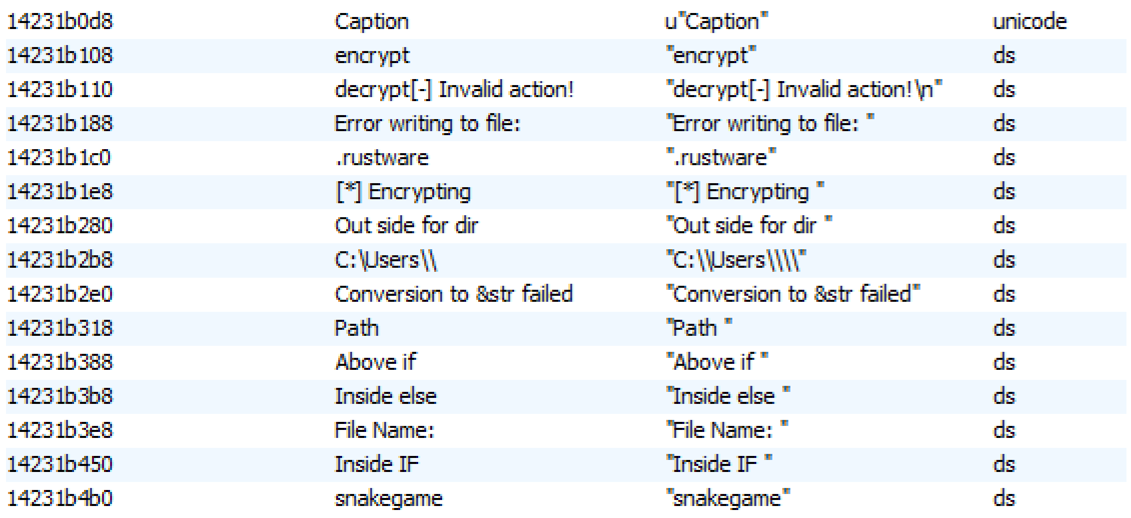
After doing that you would get a window showing you all the defined strings in the current binary. Since this is just a Static Properties analysis you could use other tools to extract the string but I usually prefer Ghidra.

Below are interesting strings that I found in the binary file that gave me an idea on what the malware was about like in Figures below



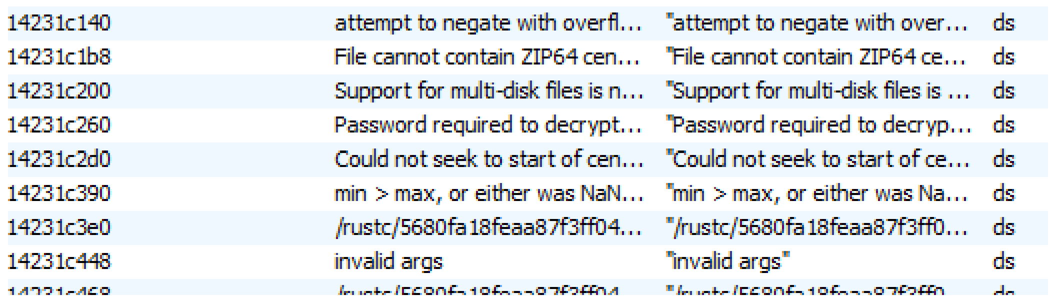
14231ad78	Software\Microsoft\Windows\...	"Software\\Microsoft\\Windo...	ds
14231ade8	Dannys'sRansomware	"Dannys'sRansomware"	ds
14231ae18	Key already there, dont do a...	"Key already there, dont do ...	ds
14231ae70	attempt to add with overflow...	"attempt to add with overflo...	ds

Figure 2



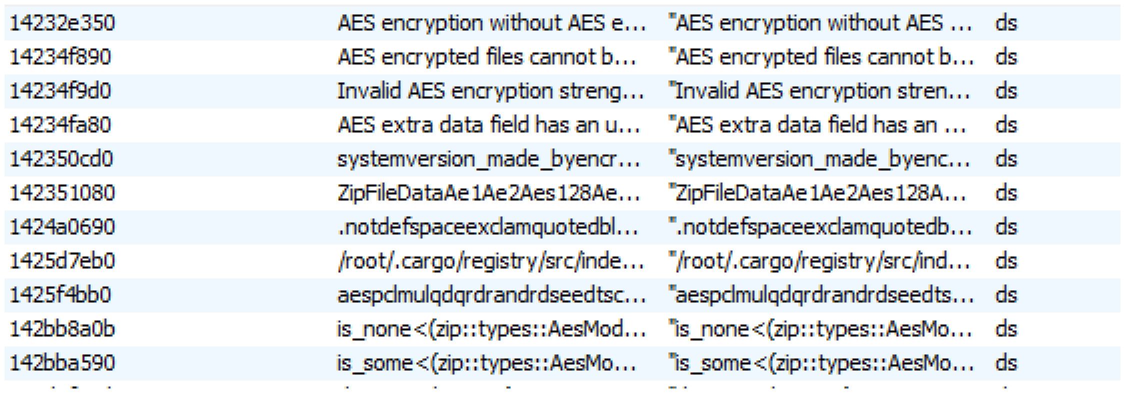
14231b0d8	Caption	u"Caption"	unicode
14231b108	encrypt	"encrypt"	ds
14231b110	decrypt[-] Invalid action!	"decrypt[-] Invalid action!\n"	ds
14231b188	Error writing to file:	"Error writing to file: "	ds
14231b1c0	.rustware	".rustware"	ds
14231b1e8	[*] Encrypting	"[*] Encrypting "	ds
14231b280	Out side for dir	"Out side for dir "	ds
14231b2b8	C:\Users\\	"C:\\Users\\\\"	ds
14231b2e0	Conversion to &str failed	"Conversion to &str failed"	ds
14231b318	Path	"Path "	ds
14231b388	Above if	"Above if "	ds
14231b3b8	Inside else	"Inside else "	ds
14231b3e8	File Name:	"File Name: "	ds
14231b450	Inside IF	"Inside IF "	ds
14231b4b0	snakegame	"snakegame"	ds

Figure 3



14231c140	attempt to negate with overfl...	"attempt to negate with over...	ds
14231c1b8	File cannot contain ZIP64 cen...	"File cannot contain ZIP64 ce...	ds
14231c200	Support for multi-disk files is n...	"Support for multi-disk files is ...	ds
14231c260	Password required to decrypt...	"Password required to decryp...	ds
14231c2d0	Could not seek to start of cen...	"Could not seek to start of ce...	ds
14231c390	min > max, or either was NaN...	"min > max, or either was Na...	ds
14231c3e0	/rustc/5680fa18feaa87f3ff04...	"/rustc/5680fa18feaa87f3ff0...	ds
14231c448	invalid args	"invalid args"	ds

Figure 4



14232e350	AES encryption without AES e...	"AES encryption without AES ...	ds
14234f890	AES encrypted files cannot b...	"AES encrypted files cannot b...	ds
14234f9d0	Invalid AES encryption streng...	"Invalid AES encryption stren...	ds
14234fa80	AES extra data field has an u...	"AES extra data field has an ...	ds
142350cd0	systemversion_made_byencr...	"systemversion_made_byenc...	ds
142351080	ZipFileDataAe1Ae2Aes128Ae...	"ZipFileDataAe1Ae2Aes128A...	ds
1424a0690	.notdefspaceexclamquotedbl...	".notdefspaceexclamquotedb...	ds
1425d7eb0	/root/.cargo/registry/src/inde...	"/root/.cargo/registry/src/ind...	ds
1425f4bb0	aespclmulqdqrdrandrdseedtsc...	"aespclmulqdqrdrandrdseedts...	ds
142bb8a0b	is_none<(zip::types::AesMod...	"is_none<(zip::types::AesMo...	ds
142bba590	is_some<(zip::types::AesMo...	"is_some<(zip::types::AesMo...	ds

Figure 5

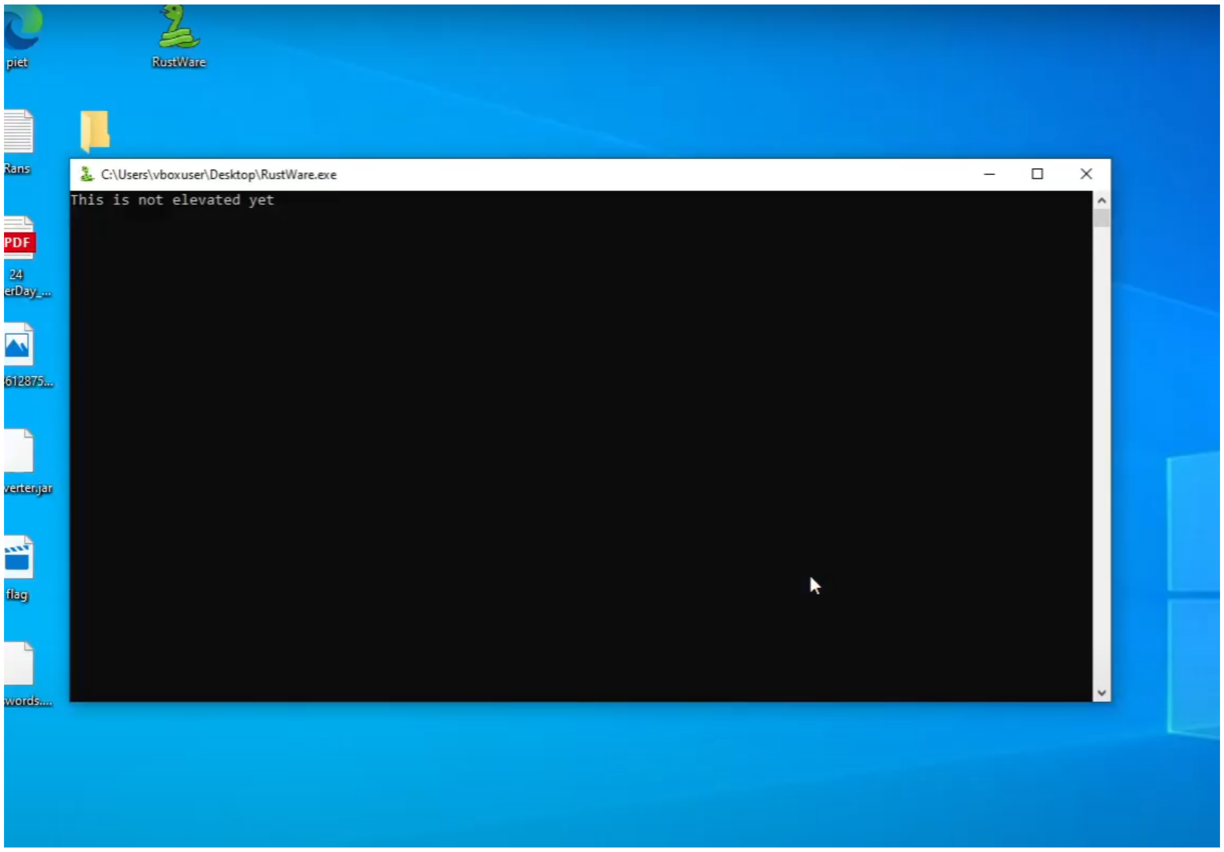
Looking at the general analysis with string one can infer that the type of Malware this windows binary files could be a ransomware type of files that deals with some type of game. It also looks like it encrypts/decrypts files with AES encryption as well as searches through the user directory on Windows. So without initially running the binary files and doing static analysis we get an idea on what the binary does very quickly with just strings.

Interactive Behavior

In our interactive behavior analysis now we can interact with the malware either through a sandbox environment online or locally to make observations about the malware behavior and take notes about it as well to try and understand what the malware is doing in real time.

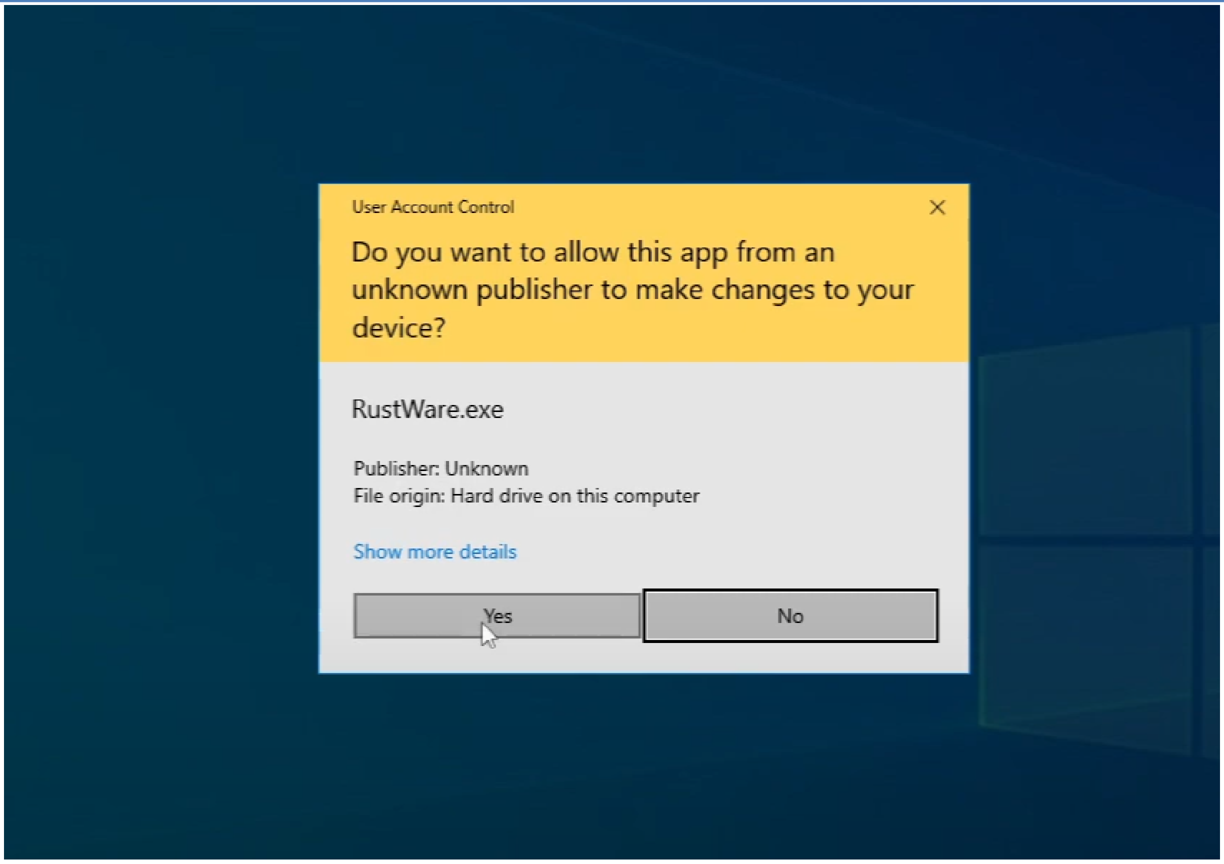
So in my case I used Virtual Box which is virtualization software meant to imitate a real computer system. This tool allows me to interact with the malware without damaging my host machine and not having to worry about the malware infecting my own personal system

Initially clicking the file we get a prompt saying this is not elevated yet



```
This is not elevated yet
```

Figure 6



User Account Control

Do you want to allow this app from an unknown publisher to make changes to your device?

RustWare.exe

Publisher: Unknown
File origin: Hard drive on this computer

Figure 7

Later on in figure 7 it ask use in general if we want this file to give us permission to elevate its privileged and if you click yes you get a snake game below in Figure 8

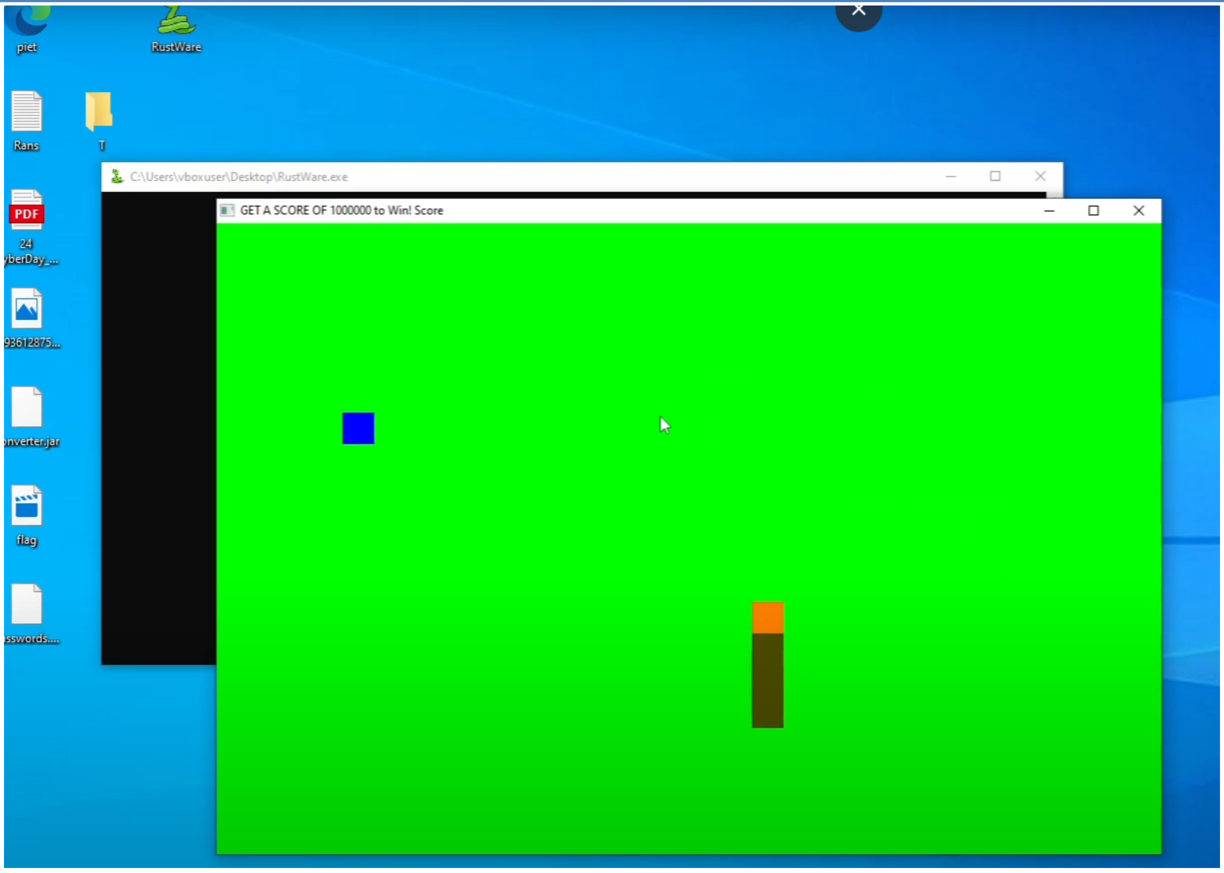
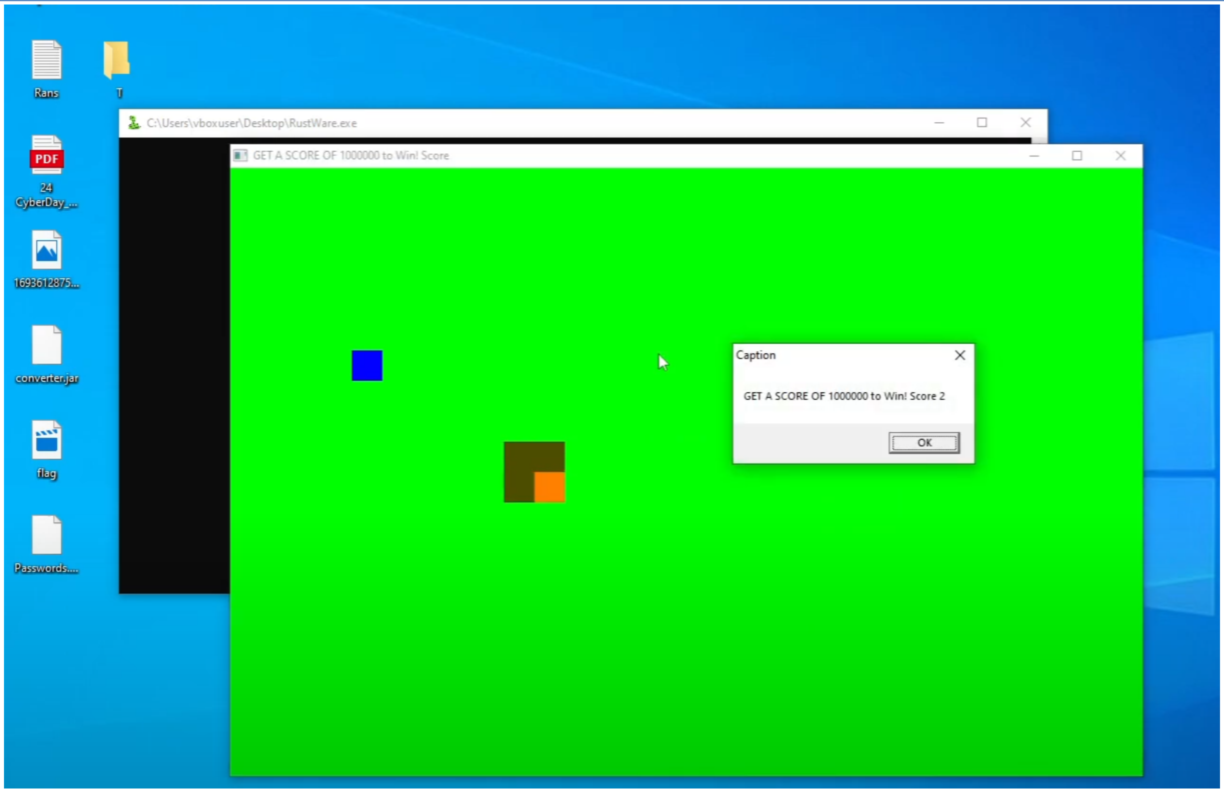
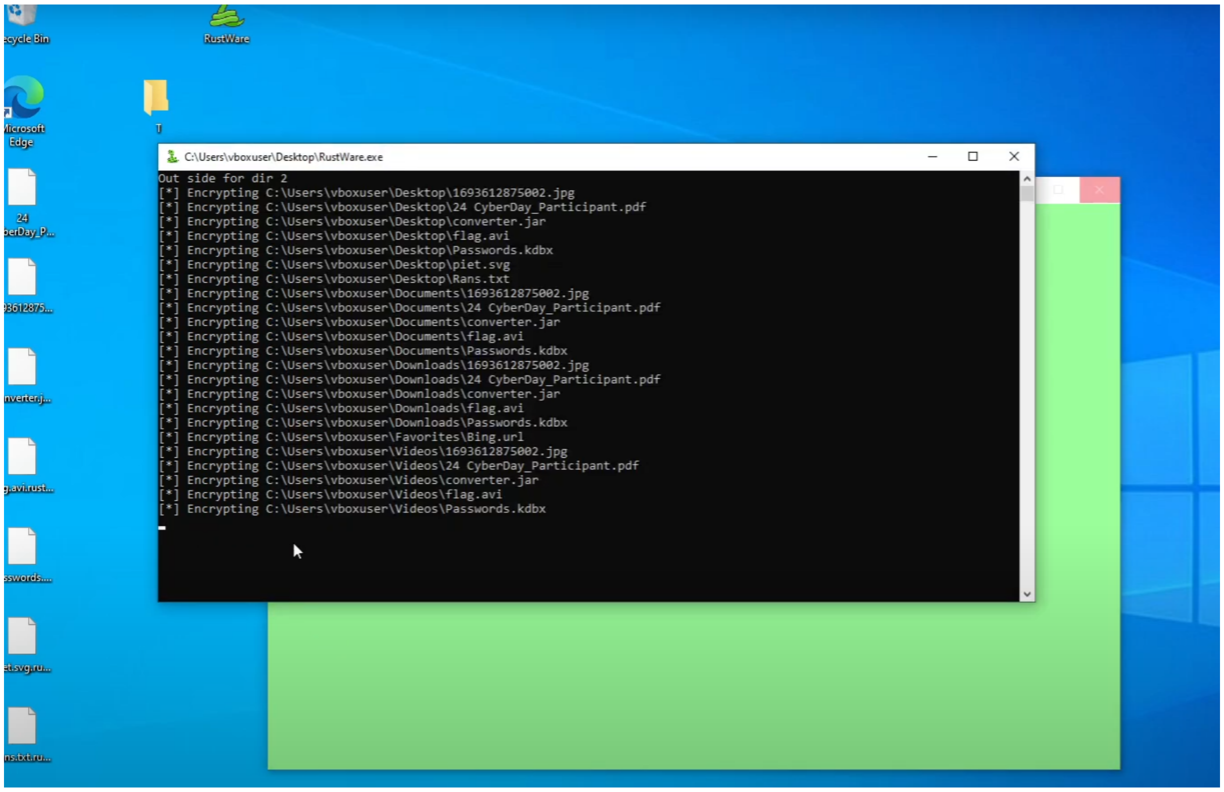

piet
RustWare
Rans
T
PDF
24
yberDay_...
93612875...
nverter.jar
flag
sswords....
C:\Users\vboxuser\Desktop\RustWare.exe
GET A SCORE OF 1000000 to Win! Score

Figure 8


Rans
T
24 CyberDay_...
1693612875...
converter.jar
flag
Passwords....
C:\Users\vboxuser\Desktop\RustWare.exe
GET A SCORE OF 1000000 to Win! Score
Caption
OK

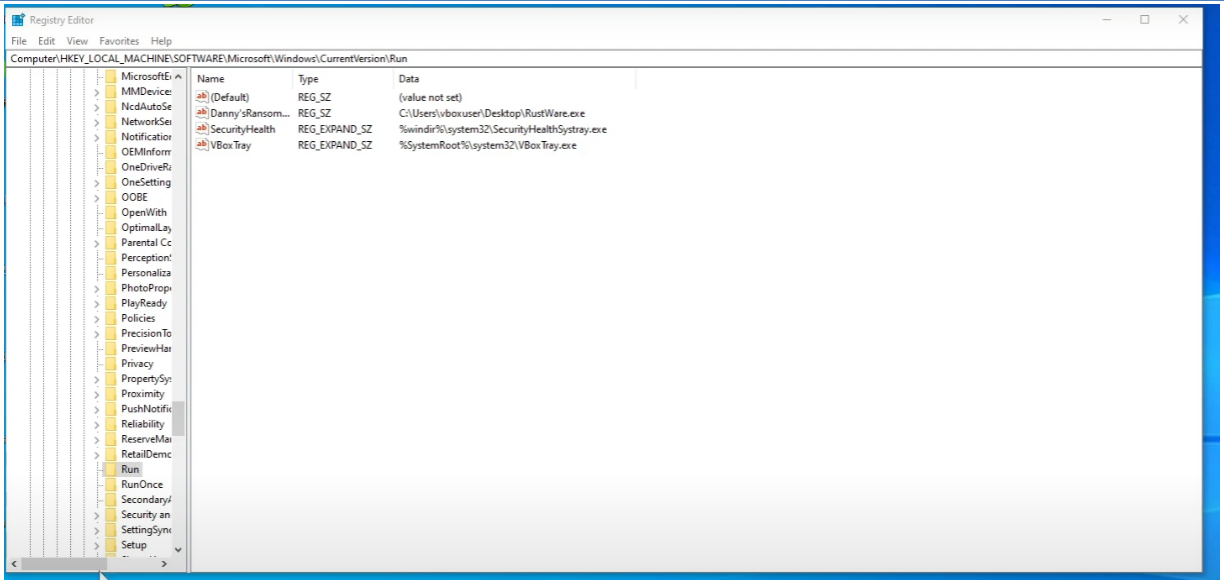
GET A SCORE OF 1000000 to Win! Score 2

Figure 9



```
Out side for dir 2
[*] Encrypting C:\Users\vboxuser\Desktop\1693612875002.jpg
[*] Encrypting C:\Users\vboxuser\Desktop\24 CyberDay_Participant.pdf
[*] Encrypting C:\Users\vboxuser\Desktop\converter.jar
[*] Encrypting C:\Users\vboxuser\Desktop\flag.avi
[*] Encrypting C:\Users\vboxuser\Desktop\Passwords.kdbx
[*] Encrypting C:\Users\vboxuser\Desktop\piet.svg
[*] Encrypting C:\Users\vboxuser\Desktop\Rans.txt
[*] Encrypting C:\Users\vboxuser\Documents\1693612875002.jpg
[*] Encrypting C:\Users\vboxuser\Documents\24 CyberDay_Participant.pdf
[*] Encrypting C:\Users\vboxuser\Documents\converter.jar
[*] Encrypting C:\Users\vboxuser\Documents\flag.avi
[*] Encrypting C:\Users\vboxuser\Documents\Passwords.kdbx
[*] Encrypting C:\Users\vboxuser\Downloads\1693612875002.jpg
[*] Encrypting C:\Users\vboxuser\Downloads\24 CyberDay_Participant.pdf
[*] Encrypting C:\Users\vboxuser\Downloads\converter.jar
[*] Encrypting C:\Users\vboxuser\Downloads\flag.avi
[*] Encrypting C:\Users\vboxuser\Downloads\Passwords.kdbx
[*] Encrypting C:\Users\vboxuser\Favorites\Bing.url
[*] Encrypting C:\Users\vboxuser\Videos\1693612875002.jpg
[*] Encrypting C:\Users\vboxuser\Videos\24 CyberDay_Participant.pdf
[*] Encrypting C:\Users\vboxuser\Videos\converter.jar
[*] Encrypting C:\Users\vboxuser\Videos\flag.avi
[*] Encrypting C:\Users\vboxuser\Videos\Passwords.kdbx
```

Figure 10



Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

Name	Type	Data
(Default)	REG_SZ	(value not set)
Danny'sRansom...	REG_SZ	C:\Users\vboxuser\Desktop\RustWare.exe
SecurityHealth	REG_EXPAND_SZ	%windir%\system32\SecurityHealthSystray.exe
VBoxTray	REG_EXPAND_SZ	%SystemRoot%\system32\VBoxTray.exe

Figure 11

Throughout the rest of the Interactive Behavior analysis we found out that if you reach a certain score in our case we only score two that means the game begins to close and the rest of the files seems to be getting encrypted. Through further analysis I just checked the registry to see what was going on and noticed that a register was added to the HKEY_LOCAL_MACHINE register which means every time Windows restart or shutdown and gets turned on again the malware sample seems to restart . Initially it seems in order to win or to decrypt the files that I currently have I need to find a way to reach that score of 10000 that was prompted out to us each time we lost.

Manual Code Reversing

This section we start dissecting the malware for the most part by using reverse engineering tools so in this part we are going to be using the full functionality of Ghidra.

While decompiling it on Ghidra we see a large amount of folders means that there are going to be a lot of functions that need to go over and files that Ghidra decompile with these malware files since it was built upon rust as you can see in Figure 12 .

So to get bypass this since we know that this Malware is ransomware we can use that information to good use later on in Figure 2 to Figure 5 we have examples of what happens when we extract the strings but what we can do is go to the instances and see where it decrypt and encrypt the file and if we can see the logic behind it we can then reverse the functionally to decrypt the file without trying to beat the game itself

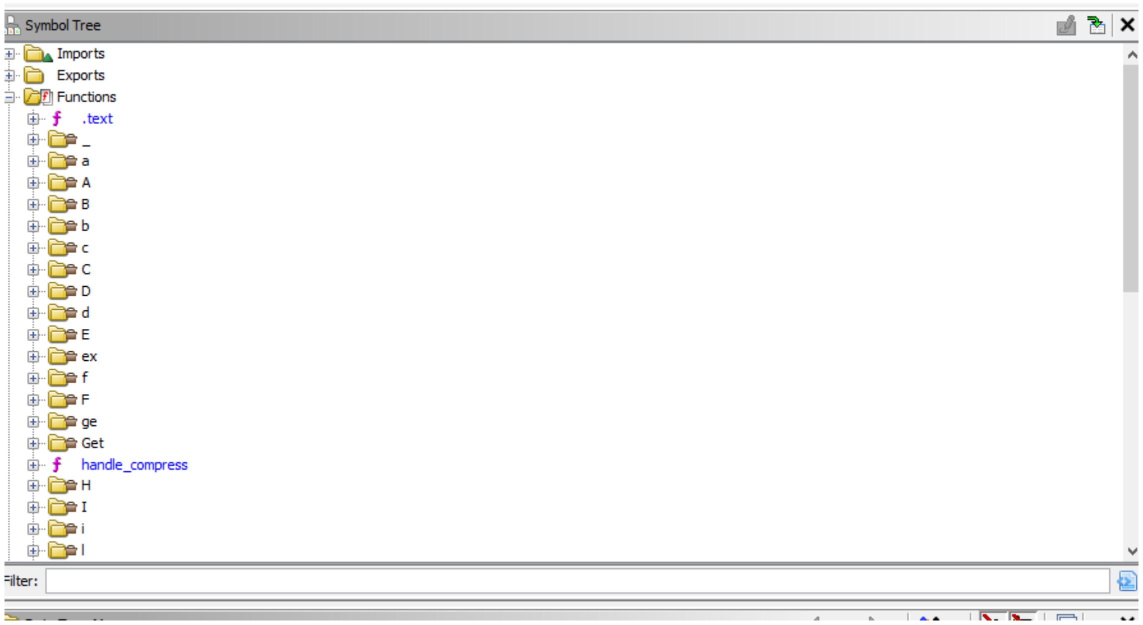
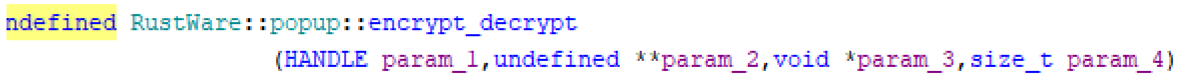


Figure 12

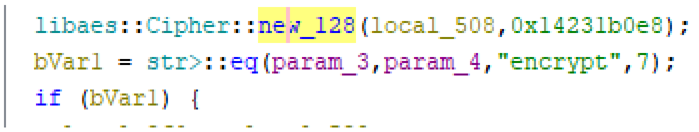
Down below in Figure 13 I was able to find the encrypt_decrypt functions located in the file. So we are getting close to reversing this function



```
ndefined RustWare::popup::encrypt_decrypt
                   (HANDLE param_1,undefined **param_2,void *param_3,size_t param_4)
```

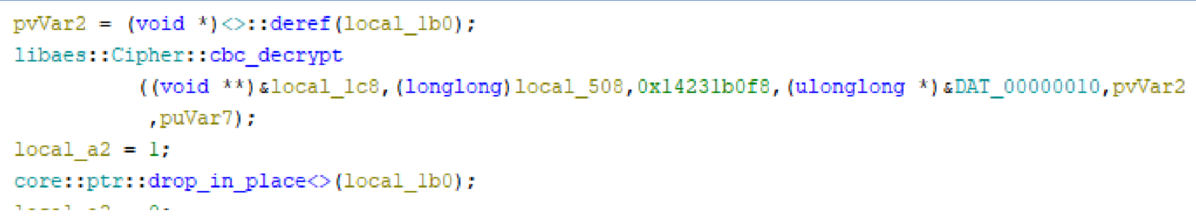
Figure 13

While looking over the function I found two functions that seems interesting to me which was new_128 and cbc_decrypt the reason being is because new_128 makes a cipher and then we later on use that exact cipher to decrypt in both figures 14 and figures 15 so I did a quick google search and found that their is example code online to encrypt and decrypt files using libaes



```
libaes::Cipher::new_128(local_508,0x14231b0e8);
bVar1 = str>::eq(param_3,param_4,"encrypt",7);
if (bVar1) {
```

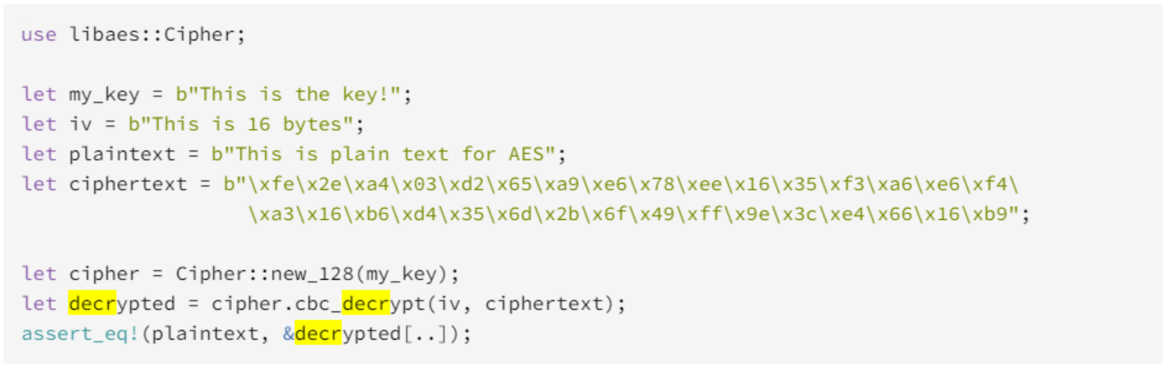
Figure 14



```
pvVar2 = (void *)<>::deref(local_1b0);
libaes::Cipher::cbc_decrypt
          ((void **)&local_1c8,(longlong)local_508,0x14231b0f8,(ulonglong *)&DAT_00000010,pvVar2
           ,puVar7);
local_a2 = 1;
core::ptr::drop_in_place<>(local_1b0);
```

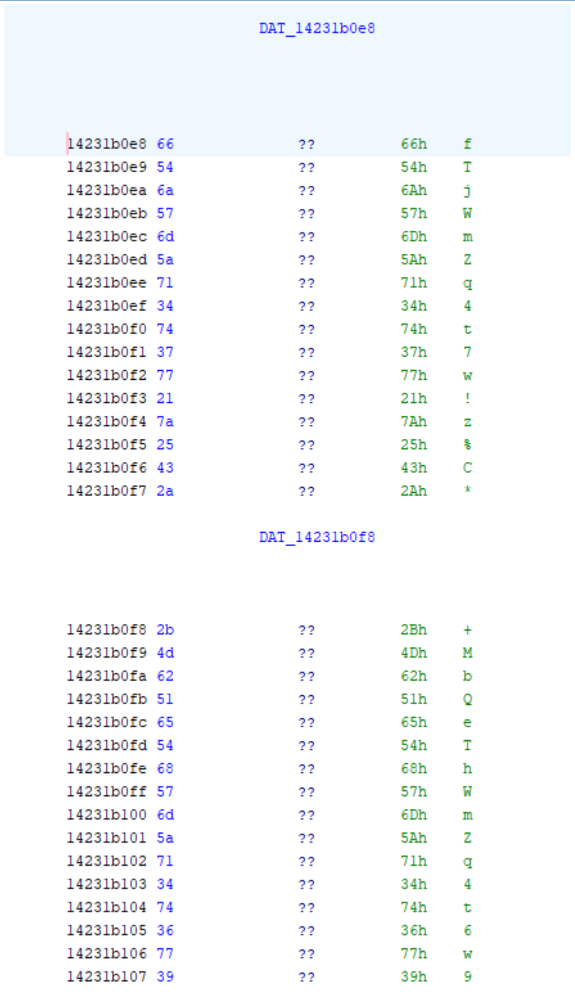
Figure 15

In figure 16 I was able to find the example to show how to decrypt the files given the plain text https://docs.rs/libaes/latest/libaes/struct.Cipher.html on this site. Looking over at the site I compared the two functions and started to look over the example and compare both the pseudo code generated and the example on the rust docs. From There on I was able to find the my_key parameter and the iv parameter inside the pseudocode like show in Figure 17.



```
use libaes::Cipher;

let my_key = b"This is the key!";
let iv = b"This is 16 bytes";
let plaintext = b"This is plain text for AES";
let ciphertext = b"\xfe\x2e\xa4\x03\xd2\x65\xa9\xe6\x78\xee\x16\x35\xf3\xa6\xe6\xf4\
                   \xa3\x16\xb6\xd4\x35\x6d\x2b\x6f\x49\xff\x9e\x3c\xe4\x66\x16\xb9";

let cipher = Cipher::new_128(my_key);
let decrypted = cipher.cbc_decrypt(iv, ciphertext);
assert_eq!(plaintext, &decrypted[..]);
```

Figure 16

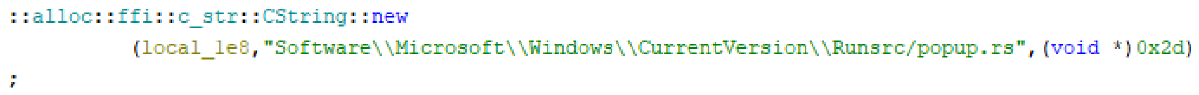


```
                 DAT_14231b0e8

14231b0e8 66         ??        66h    f
14231b0e9 54         ??        54h    T
14231b0ea 6a         ??        6Ah    j
14231b0eb 57         ??        57h    W
14231b0ec 6d         ??        6Dh    m
14231b0ed 5a         ??        5Ah    Z
14231b0ee 71         ??        71h    q
14231b0ef 34         ??        34h    4
14231b0f0 74         ??        74h    t
14231b0f1 37         ??        37h    7
14231b0f2 77         ??        77h    w
14231b0f3 21         ??        21h    !
14231b0f4 7a         ??        7Ah    z
14231b0f5 25         ??        25h    %
14231b0f6 43         ??        43h    C
14231b0f7 2a         ??        2Ah    *

                 DAT_14231b0f8

14231b0f8 2b         ??        2Bh    +
14231b0f9 4d         ??        4Dh    M
14231b0fa 62         ??        62h    b
14231b0fb 51         ??        51h    Q
14231b0fc 65         ??        65h    e
14231b0fd 54         ??        54h    T
14231b0fe 68         ??        68h    h
14231b0ff 57         ??        57h    W
14231b100 6d         ??        6Dh    m
14231b101 5a         ??        5Ah    Z
14231b102 71         ??        71h    q
14231b103 34         ??        34h    4
14231b104 74         ??        74h    t
14231b105 36         ??        36h    6
14231b106 77         ??        77h    w
14231b107 39         ??        39h    9
```

Figure 17

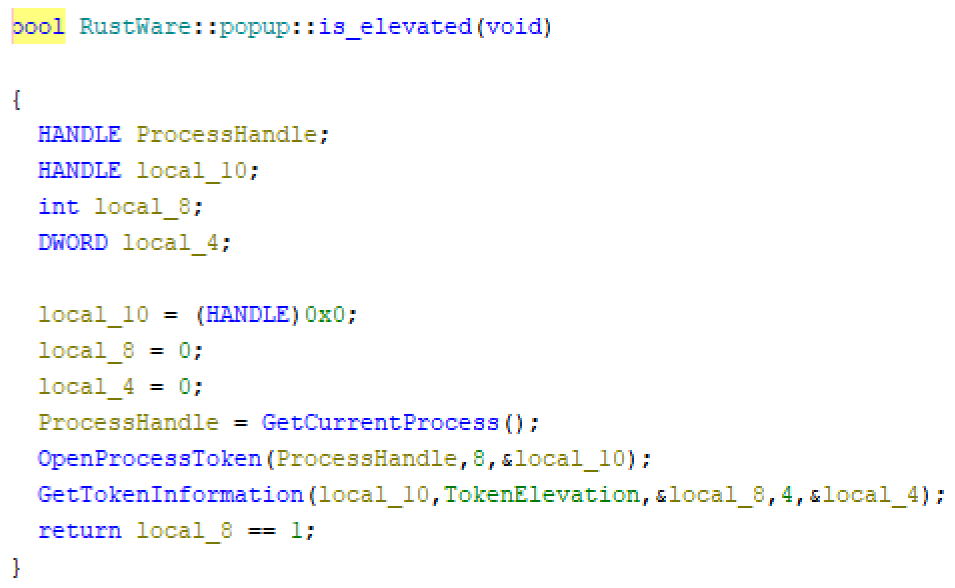
So that means that since we have both the key and iv we do not need to play the game and beat the game to decrypt a file. With this information given to use all we need to do is make our own script using rust to decrypt the file.

Alongside those findings that I stated earlier I also found the entry where it saves the our code into the registers to when on boot up the malware runs again. Also down below on Figure 19 and Figure 20. We can see the program is checking if it has elevated privileges and the folder it seems to be attacking in the system.



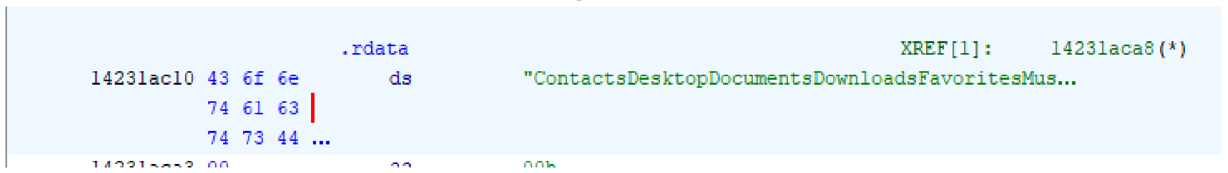
```
::alloc::ffi::c_str::CString::new
          (local_1e8,"Software\\Microsoft\\Windows\\CurrentVersion\\Runsrc/popup.rs",(void *)0x2d)
;
```

Figure 18



```
bool RustWare::popup::is_elevated(void)

{
  HANDLE ProcessHandle;
  HANDLE local_10;
  int local_8;
  DWORD local_4;

  local_10 = (HANDLE)0x0;
  local_8 = 0;
  local_4 = 0;
  ProcessHandle = GetCurrentProcess();
  OpenProcessToken(ProcessHandle,8,&local_10);
  GetTokenInformation(local_10,TokenElevation,&local_8,4,&local_4);
  return local_8 == 1;
}
```

Figure 19



```
                        .rdata                                          XREF[1]:     1423laca8(*)
    1423lac10 43 6f 6e       ds          "ContactsDesktopDocumentsDownloadsFavoritesMus...
              74 61 63
              74 73 44 ...
```

Figure 20

Attachments

- Virtus Total Link: https://www.virustotal.com/gui/file/ca9d23d4be4fc5c1bae0f57b681c462d31d7b817627a966981c329c84813102e/detection
- Ghidra: https://github.com/NationalSecurityAgency/ghidra
- Virtual Box https://www.virtualbox.org/