Katsarakis Zaxarias
AM: 2898
Email: csd2898@csd.uoc.gr

HY 457 Report

Reverse Engineering report

1)

Command executed :

file -f dbb4fdbba0f4bb8b441c1d610260bbc8.VBR

File mal

Ransomware file option

```
zaxariaskatsarakis@Zaxariass-MacBook-Air rev_lab_hw % file mal
mal: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=578801e1d085d00d54687df9b
75e8a1854c87, for GNU/Linux 3.2.0, with debug_info, not stripped
```

ELF 64-biy LSB pie executable is targeting x86_64

Its a vbr file with the note

We are VBR(Very Bad Ransomware). We've encrypted your data! Send 25.6BTC in under 24 hours to the following address in order to decrypt your data: 9cdae1ba850148d625241ec52275ee5f

Inside the encrypted file is the data of the company

And they need to pay so they get the reverse code to decrypt it

```
zaxariaskatsarakis@Zaxariass-MacBook-Air rev_lab_hw % file -f dbb4fdbba0f4bb8b441c1d610260bbc8.VBR
VULNUKEJ        :
                        cannot open `VULNUKEJ    ' (No such file or directory)

"@QNGJTULT     N"":
                     cannot open `
                                 "@QNGJTULT    N"'' (No such file or directory)
:
                cannot open `' (No such file or directory)
We are VBR(Very Bad Ransomware). We've encrypted your data! Send 25.6BTC in under 24 hours to the following address in order to decrypt your data: 9cdae1ba850148d625
241ec52275ee5f: cannot open `We are VBR(Very Bad Ransomware). We've encrypted your data! Send 25.6BTC in under 24 hours to the following address in order to decrypt
your data: 9cdae1ba850148d625241ec52275ee5f' (No such file or directory)
```

Screenshot of the command and the note that they left behind

Analysis

A.

The ransomware actually shows that double calls every function as shown below

```
[0x00001348]
   ;-- payload:
 dbg.payload ();
 ; var char [256] buffer @ rbp-0x110
 ; var long int fsize @ rbp-0x10
 ; var FILE *fptr @ rbp-0x8
 push rbp                          ; void payload();
 mov rbp, rsp
 sub rsp, 0x110
 lea rax, [0x000020ec]
 mov rsi, rax                      ; const char *mode
 lea rax, str.mypasswords.txt      ; 0x20ef
 mov rdi, rax                      ; const char *filename
 call fopen                        ; sym.imp.fopen ; FILE *fopen(const char *filename, const char...
 mov qword [fptr], rax
 cmp qword [fptr], 0
 jne 0x1381
```

So the analyst doesn't know where the function actually begins and if the variables are actually declared or not .

If you see it in the Ghidra decompiler you will see that the actual second call is comment out .

In addition you can see that the variables declared at first they are actually used in the Code below the second call so they are already declared

B.

The main ransomware function is payload();

```
undefined8 main(int argc, char **argv)
{
    int64_t iVar1;
    undefined8 in_R8;
    undefined8 in_R9;
    char **var_10h;
    int var_4h;

    // int main(int argc,char ** argv);
    iVar1 = ptrace(0, 0, 0, 0, in_R8, in_R9, argv);
    if (iVar1 == -1) {
        fwrite("Stop debugging me!\n", 1, 0x13, _stdout);
    // WARNING: Subroutine does not return
        exit(0xffffffff);
    }
    payload();
    return 0;
}
```

The ptrace is used to check if the malware is getting dbuged so it checks where the program is being run.

And the malware uses payload to manipulates any files of the system

C.

```
0x00001348      push rbp            ; void payload();
0x00001349      mov rbp, rsp
0x0000134c      sub rsp, 0x110
0x00001353      lea rax, [0x000020ec]
0x0000135a      mov rsi, rax        ; const char *mode
0x0000135d      lea rax, str.mypasswords.txt ; 0x20ef
0x00001364      mov rdi, rax        ; const char *filename
0x00001367      call fopen          ; sym.imp.fopen ; FILE *fopen(const char *filename, const char *mode)
```

Function payload is using mypasswords.txt and gets its content using fseek sys call

D.

Payload function does the below things:

First of all opens the file my passwords.txt with fopen and the content of address 0x0000020ec

```
0x00001348      push rbp            ; void payload();
0x00001349      mov rbp, rsp
0x0000134c      sub rsp, 0x110
0x00001353      lea rax, [0x000020ec]
0x0000135a      mov rsi, rax        ; const char *mode
0x0000135d      lea rax, str.mypasswords.txt ; 0x20ef
0x00001364      mov rdi, rax        ; const char *filename
0x00001367      call fopen          ; sym.imp.fopen ; FILE *fopen(const char *filename, const char *mode)
0x0000136c      mov qword [fptr], rax
```

After fseeks the file From SEEK_BEG (0) to SEEK_END(2)

```
0x00001385      mov edx, 2          ; int whence
0x0000138a      mov esi, 0          ; long offset
0x0000138f      mov rdi, rax        ; FILE *stream
0x00001392      call fseek          ; sym.imp.fseek ; int fseek(FILE *stream, long offset, int whence)
0x00001397      mov rax, qword [fptr]
0x0000139b      mov rdi, rax        ; FILE *stream
```

After finds the size of the file with ftell and saves it to arg2

And saves the whole file to the buffer one by one

And encrypts the buffer using two for arguments for each char in the text

After finishing the encryptions puts the note we saw at the end of the buffer

And writes the encrypted messages in

dbb4fdbba0f4bb8b441c1d610260bbc8.VBR file that they set us back

```
fseek(iVar2, 0, 2);
arg2 = ftell(iVar2);
fseek(iVar2, 0, 0);
fread(buffer, arg2, 1, iVar2);
do_encrypt(buffer, arg2 & 0xffffffff);
iVar1 = put_message(buffer, arg2);
write_encrypted(buffer, (void *)(arg2 + (int64_t)iVar1));
fclose(iVar2);
return;
```

E

do_encrypt(buffer,sizeoffile-1);

Lets take the code simplify it:

```
void do_encrypt(void *arg1, long arg2)
{
    int blen;
    char *buffer;
    int i;
    int64_t var_4h;

    // void do_encrypt(char * buffer,int blen);
    for (var_4h._0_4_ = 0; (int32_t)var_4h < (int32_t)arg2; var_4h._0_4_ = (int32_t)var_4h + 1) {
        *(char *)((int64_t)arg1 + (int64_t)(int32_t)var_4h) = *(char *)((int64_t)arg1 + (int64_t)(int32_t)var_4h) + -4;
    }
    for (i = 0; i < (int32_t)arg2; i = i + 1) {
        *(uint8_t *)((int64_t)arg1 + (int64_t)i) = *(uint8_t *)((int64_t)arg1 + (int64_t)i) ^ 0x24;
    }
    return;
}
```

Long long var4h;

for(var4h = 0; int var4h < sizeofifile -1; var4h ++){

(char)(long long) arg1 + var4_h = *(char*) ((long long) arg1 + var4_h) -4;

}

This for loop takes every char and decreases by 4

And the next for

for (i = 0; i < (int32_t)arg2; i = i + 1) {

*(uint8_t *)((int64_t)arg1 + (int64_t)i) = *(uint8_t *)((int64_t)arg1 + (int64_t)i) ^ 0x24;

}

The second for takes each character of the buffer and xors it with 0x24(36 in decimal) and returns the whole encrypted buffer back to the payload function

F.

The algorithm is reversible .

Each character that is encode is -4 and xor with 0x24 .

So we have to do the exact opposite of the encoding doing in VBR file we have.

In detail;

First we open and read the file.

Then we take each character of the file and XOR it with 0x24 so get the original character that was used .

After instead of decreasing 4 like the ransomware does we add 4 and we get the original character used and we store all that in one file

At the end of the file we have the message that they putted in.

G,

They append this Message that we show before when we cat the actual vbr file

We are VBR(Very Bad Ransomware). We\'ve encrypted your data! Send 25.6BTC in under 24 hours to the following address in order to decrypt your data: 9cdae1ba850148d625241ec52275ee5f

Two times actually

```
while( true ) {
    uVar2 = strlen(
                    "We are VBR(Very Bad Ransomware). We\'ve encrypted your data! Send 25.6BTC in under 24 hours to the following a
                    );
    if (uVar2 <= (uint64_t)(int64_t)i) break;
    *(char *)(fpoint + (int64_t)arg1) =
         "We are VBR(Very Bad Ransomware). We\'ve encrypted your data! Send 25.6BTC in under 24 hours to the following address in
         [i];
    i = i + 1;
    fpoint = fpoint + 1;
}
iVar1 = strlen(
                "We are VBR(Very Bad Ransomware). We\'ve encrypted your data! Send 25.6BTC in under 24 hours to the following addre
                );
return iVar1 + 2;
```

H.

The new name of the encrypted file dbb4fdbba0f4bb8b441c1d610260bbc8.VBR

```
void write_encrypted(void *arg1, void *arg2)
{
    int64_t iVar1;
    long int flen;
    char *buffer;
    FILE *fptr;

    // void write_encrypted(char * buffer,long int flen);
    iVar1 = fopen("dbb4fdbba0f4bb8b441c1d610260bbc8.VBR", 0x2008);
    if (iVar1 != 0) {
        fwrite(arg1, arg2, 1, iVar1);
        fclose(iVar1);
    }
    return;
}
```

Reversed Code

Wer reversed the code of encryption that the malware used:

The code is bellow and we also add the actual program in c to Out report

We read every char of the .VBR file and reverse it using the idea above and actually print the real content before the encryption.

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    FILE *read, *write;
    int i , k,len;
    char *buffer;
    char *decode;

    read = fopen("dbb4fdbba0f4bb8b441c1d610260bbc8.VBR","r");
    if(read == NULL){
        printf("Failed to read the VBR \n");
        return 0;
    }
    fseek(read, 0, SEEK_END);
    len = ftell(read);
    printf("Size of VBR %d \n", len);
    buffer = malloc(len * sizeof(char));
    fseek(read,0,SEEK_SET);
    fread(buffer, len , 1 , read);
    printf("The encrypted content is: \n %s \n",buffer);
    for (int i = 0; i < len; i++)
    {
        buffer[i] = buffer[i] ^ 0x24;
        buffer[i] = buffer[i] + 4;
        if(buffer[i] == '\n'){
            if (buffer[i+1] == '\n')
            {
                decode = malloc(i * sizeof(char));
                memcpy(decode,buffer,i);
                break;
            }

        }
    }
    printf("\nThe  content  after decryption is: \n %s
\n",decode);

}
```