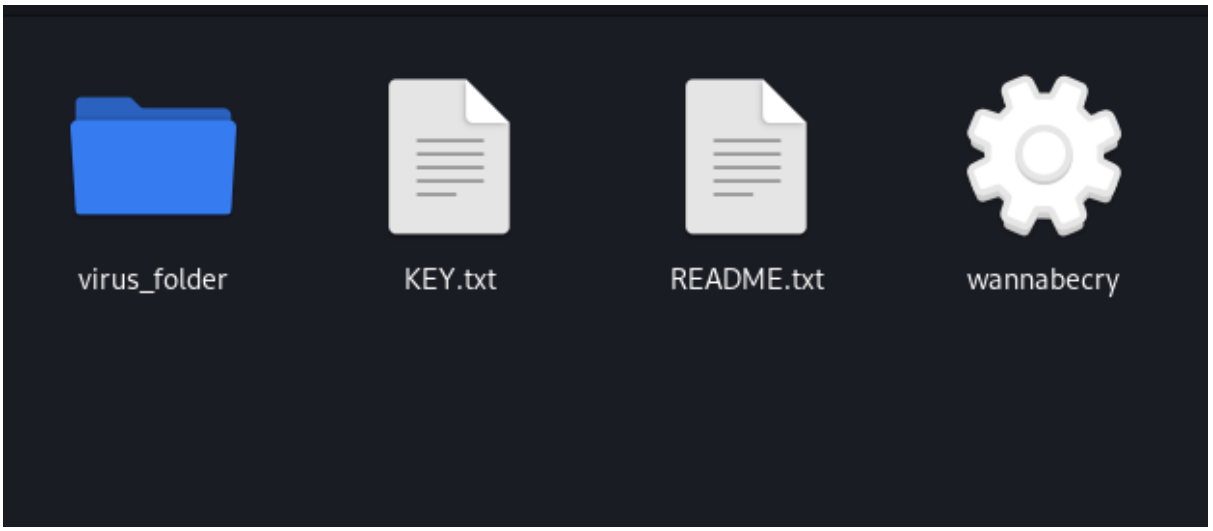
Reversing WannabeCry

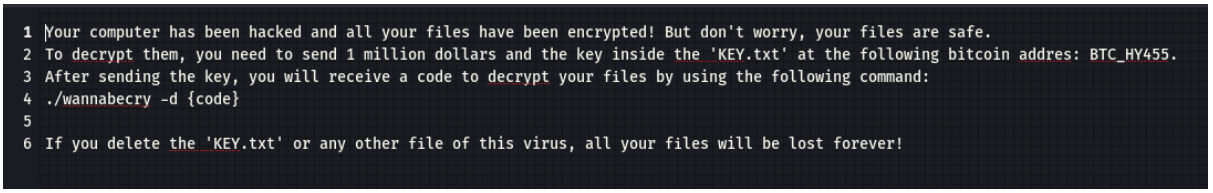
First i created a directory test and pasted wannacry executable.
Then make a directory test/virus_folder and paste this assignments pdf in there.
Then we run the ransomware and our files in the virus_folder are encrypted and we get 2 new txt files one contains a key and the other a message that says i need to send money to bitcoin address accompanied by the key.

test directory



KEY.txt

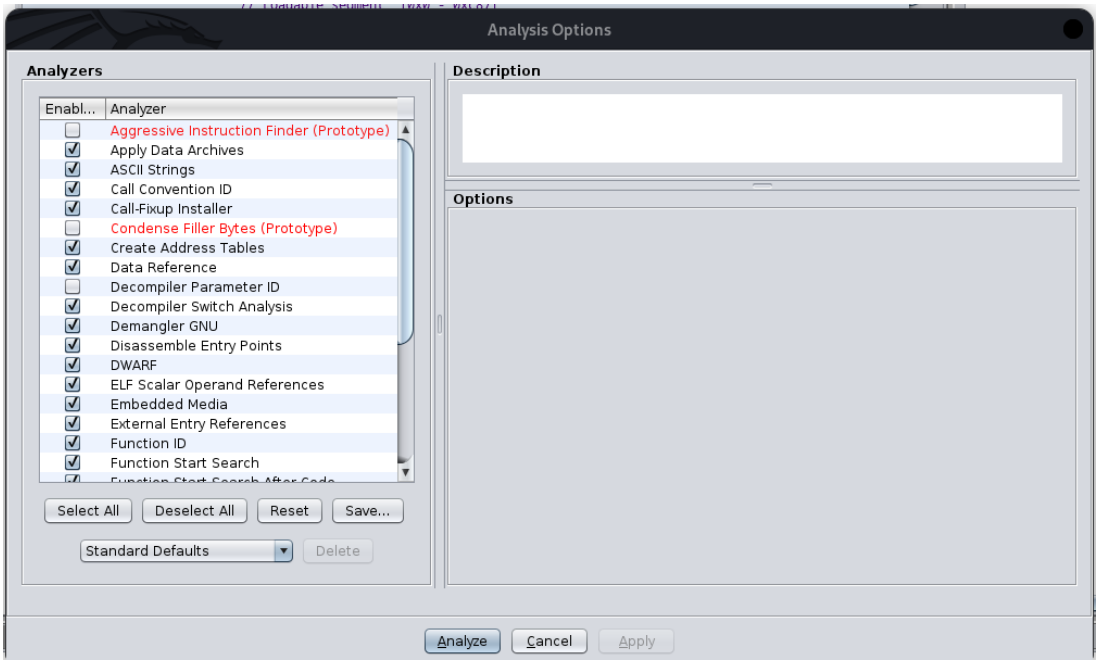
README.TXT



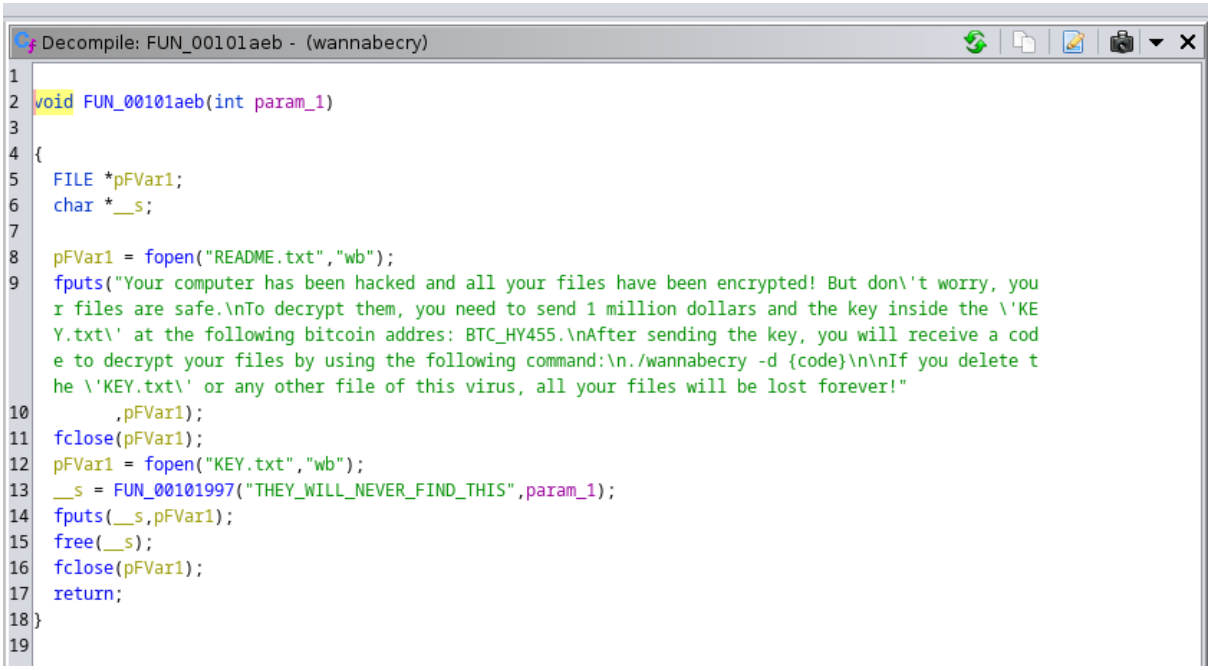
Now let's try to decrypt bobs files.
First we create a new ghidra project and import wannabecry binary file.

we see that ghidra detected 65 functions along with other info that might be useful.

Now we double click our binary in project view and we tell ghidra to analyze it .



now the binary is analyzed. After searching for a bit in the functions we find the function that creates the README.TXT file :



This function simply creates a file named README.rxt and then appends the message that you can find in it later but then in line 13 we can see that it also prepares something to append to the file KEY.txt that is suspicious. Let's check that function.

```
pFVar1 = fopen("KEY.txt","wb");
__s = FUN_00101997("THEY_WILL_NEVER_FIND_THIS",param_1);
fputs(__s,pFVar1);
```

Ok after searching and playing around with that function we can beautify the code and we slowly see that this is the function that creates the key for the key.txt file but it uses one more function in it:

Decompile: CREATE_KEY - (wannabecry)

```
1
2  char * CREATE_KEY(char *ARG1,int param_2)
3
4  {
5    int ARG1LENtmp;
6    int iVar1;
7    size_t ARG1LEN;
8    char *ARG1_STR_MEM_ALOC;
9    int COUNT;
10
11   ARG1LEN = strlen(ARG1);
12   ARG1LENtmp = (int)ARG1LEN;
13   ARG1_STR_MEM_ALOC = (char *)malloc((long)(ARG1LENtmp + 1));
14   COUNT = 0;
15   while( true ) {
16     if (ARG1LENtmp <= COUNT) {
17       ARG1_STR_MEM_ALOC[ARG1LENtmp] = '\0';
18       return ARG1_STR_MEM_ALOC;
19     }
20     iVar1 = FUN_001014a9(ARG1[COUNT]);
21     if (iVar1 == -1) break;
22     ARG1_STR_MEM_ALOC[COUNT] =
23          " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuv
           wxyz{|}~"
24         [(param_2 + iVar1) % 0x5f];
25     COUNT = COUNT + 1;
26   }
27   return "INVALID";
28 }
29
```

Remember the function takes as char *ARG1 **"THEY_WILL_NEVER_FIND_THIS"** .

we can see that the this function uses another function for its char of ARG1
that does this :

Decompile: FIND_CHAR_NUM_FROM_0-94 - (wannabecry)

```
1
2  int FIND_CHAR_NUM_FROM_0-94(char one_char_of_key)
3
4  {
5    int count;
6
7    count = 0;
8    while( true ) {
9      if (94 < count) {
10       return -1;
11     }
12     if (one_char_of_key ==
13         " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvw
           xyz{|}~"
14         [count]) break;
15     count = count + 1;
16   }
17   return count;
18 }
19
```

DISCLAIMER: I changed stuff so it can be readable . Basically this function takes one by one the chars of char *ARG1 **"THEY_WILL_NEVER_FIND_THIS"** and returns in which index of the sequence
" !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvw xyz{|}~"
each char is.

Now lets get back to the CREATE_KEY function.

```
20    index_of_char_in_sequence = FIND_CHAR_NUM_FROM_0-94(ARG1[COUNT]);
21    if (index_of_char_in_sequence == -1) break;
22    ARG1_STR_MEM_ALOC[COUNT] =
23         " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuv
      wxyz{|}~"
24         [(param_2 + index_of_char_in_sequence) % 95];
```

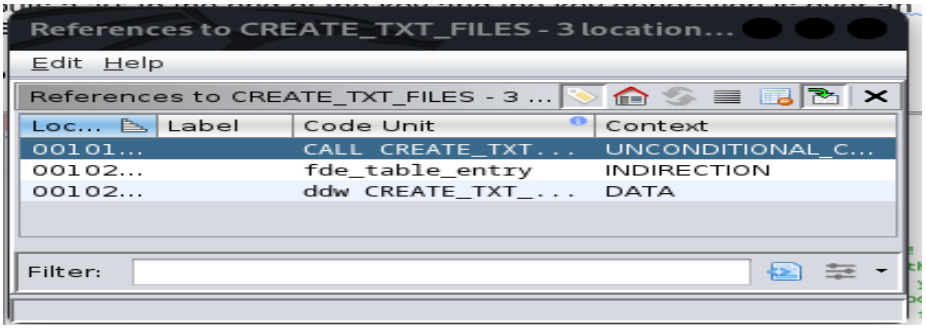
everything happens here for the key to be created :
every char of the key is
“!\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuv wxyz{|}~" [(param_2 + index_of_char_in_sequence) % 95]

where **param_2** is an int
where **index_of_char_in_sequence** is explained above .
These 2 act as an index for the sequence to select the char to be added to the key[COUNT] but modded with 95 so it can be in range.
When count exceeds or equals the length of the **"THEY_WILL_NEVER_FIND_THIS"** string it puts a ‘\0’ to the end of the key and the key generation is over an is appended inside the **KEY.txt** file.

Now we are left with this :

```
1
2  void CREATE_TXT_FILES(int param_1)
3
4  {
5    FILE *pFVar1;
6    char *KEYSTR;
7
8    pFVar1 = fopen("README.txt","wb");
9    fputs("Your computer has been hacked and all your files have been encrypted! But don\'t worry, you
     r files are safe.\nTo decrypt them, you need to send 1 million dollars and the key inside the \'KE
     Y.txt\' at the following bitcoin addres: BTC_HY455.\nAfter sending the key, you will receive a cod
     e to decrypt your files by using the following command:\n./wannabecry -d {code}\n\nIf you delete t
     he \'KEY.txt\' or any other file of this virus, all your files will be lost forever!"
10         ,pFVar1);
11   fclose(pFVar1);
12   pFVar1 = fopen("KEY.txt","wb");
13   KEYSTR = CREATE_KEY("THEY_WILL_NEVER_FIND_THIS",param_1);
14   fputs(KEYSTR,pFVar1);
15   free(KEYSTR);
16   fclose(pFVar1);
17   return;
18 }
```

We know what this function does. But we don't know where it is called and how the integer param_1 takes values . Let’s find out more.
Use the find references mechanism of ghidra (right click on the signature).we find this :



click on it and it takes us here :

```
1
2  undefined8 FUN_00101d09(int param_1,long param_2)
3
4  {
5    bool bVar1;
6    uint uVar2;
7    int iVar3;
8    undefined7 extraout_var;
9    time_t tVar4;
10   FILE *pFVar5;
11
12   bVar1 = FUN_00101b98();
13   if ((int)CONCAT71(extraout_var,bVar1) == 0) {
14     tVar4 = time((time_t *)0x0);
15     srand((uint)tVar4);
16     uVar2 = rand();
17     iVar3 = (int)((uVar2 ^ (int)uVar2 >> 0x1f) - ((int)uVar2 >> 0x1f)) % 0x5f;
18     puts("Encrypting files:");
19     FUN_00101907(iVar3);
20     CREATE_TXT_FILES(iVar3);
21     puts("Your computer has been infected. Check the generated readme file.");
22   }
23   else if (param_1 == 2) {
24     iVar3 = atoi(*(char **)(param_2 + 8));
25     pFVar5 = FUN_00101bdc(iVar3);
26     if ((int)pFVar5 == 0) {
27       puts("Invalid code! Did you pay the money or are you trying random stuff?");
28     }
29     else {
30       puts("Decrypting files:");
31       FUN_00101907(iVar3);
32       remove("KEY.txt");
33       puts("You did it! Your files are now decrypted!");
34     }
35   }
36   else {
37     puts("The only available command is ./wannabecry {code}.");
38   }
39   return 0;
40 }
```

After taking a look to this we can see some interesting things

1. Line 19,31 is the function that does the encryption/decryption.

Let’s beautify the code a bit and take a look to these Functions:

```
1
2  undefined8 FUN_00101d09(int param_1,long param_2)
3
4  {
5    bool bVar1;
6    uint randomUint;
7    int encryption/decryption_KEY;
8    undefined7 extraout_var;
9    time_t tVar2;
10   FILE *pFVar3;
11
12   bVar1 = FUN_00101b98();
13   if ((int)CONCAT71(extraout_var,bVar1) == 0) {
14     tVar2 = time((time_t *)0x0);
15     srand((uint)tVar2);
16     randomUint = rand();
17     encryption/decryption_KEY =
18          (int)((randomUint ^ (int)randomUint >> 31) - ((int)randomUint >> 31)) % 95;
19     puts("Encrypting files:");
20     encryption/decryption(encryption/decryption_KEY);
21     CREATE_TXT_FILES(encryption/decryption_KEY);
22     puts("Your computer has been infected. Check the generated readme file.");
23   }
24   else if (param_1 == 2) {
25     encryption/decryption_KEY = atoi(*(char **)(param_2 + 8));
26     pFVar3 = FUN_00101bdc(encryption/decryption_KEY);
27     if ((int)pFVar3 == 0) {
28       puts("Invalid code! Did you pay the money or are you trying random stuff?");
29     }
30     else {
31       puts("Decrypting files:");
32       encryption/decryption(encryption/decryption_KEY);
33       remove("KEY.txt");
34       puts("You did it! Your files are now decrypted!");
35     }
36   }
37   else {
38     puts("The only available command is ./wannabecry {code}.");
39   }
40   return 0;
41 }
```

(**figure 1**)

we can see that at line 26 this function returns an integer that is then compared to 0 if false (not zero) then it decrypts the files we can also see that the key that this function takes is being generated from param_2…. let's take a closer look at this function.

```
1
2  FILE * FUN_00101bdc(int param_1)
3
4  {
5    int iVar1;
6    FILE *__stream;
7    long lVar2;
8    char *__s;
9    size_t sVar3;
10   size_t sVar4;
11   char *__s1;
12
13   __stream = fopen("KEY.txt","r");
14   if (__stream != (FILE *)0x0) {
15     fseek(__stream,0,2);
16     lVar2 = ftell(__stream);
17     rewind(__stream);
18     __s = (char *)malloc((long)(int)lVar2);
19     fread(__s,(long)(int)lVar2,1,__stream);
20     fclose(__stream);
21     sVar3 = strlen(__s);
22     sVar4 = strlen("THEY_WILL_NEVER_FIND_THIS");
23     if (sVar3 == sVar4) {
24       __s1 = (char *)FUN_00101a7c(__s,param_1);
25       iVar1 = strcmp(__s1,"THEY_WILL_NEVER_FIND_THIS");
26       free(__s);
27       free(__s1);
28       __stream = (FILE *)(ulong)(iVar1 == 0);
29     }
30     else {
31       free(__s);
32       __stream = (FILE *)0x0;
33     }
34   }
35   return __stream;
36 }
37
```

After beautifying the code we come to this .

```
1
2  FILE * FUN_00101bdc(int decryption_key)
3
4  {
5    int iVar1;
6    FILE *key_txt;
7    long filepos;
8    char *__s;
9    size_t s_len;
10   size_t find_this_len;
11   char *__s1;
12
13   key_txt = fopen("KEY.txt","r");
14   if (key_txt != (FILE *)0x0) {
15                     /* find file length */
16     fseek(key_txt,0,2);
17     filepos = ftell(key_txt);
18     rewind(key_txt);
19     __s = (char *)malloc((long)(int)filepos);
20                     /* read to __s the contents of the file */
21     fread(__s,(long)(int)filepos,1,key_txt);
22     fclose(key_txt);
23     s_len = strlen(__s);
24     find_this_len = strlen("THEY_WILL_NEVER_FIND_THIS");
25     if (s_len == find_this_len) {
26                     /* __s1 = (char *)CALL_CREATE_KEY(__s,decryption_key); */
27       CALL_CREATE_KEY(__s,decryption_key);
28       iVar1 = strcmp(__s1,"THEY_WILL_NEVER_FIND_THIS");
29       free(__s);
30       free(__s1);
31       key_txt = (FILE *)(ulong)(iVar1 == 0);
32     }
33     else {
34       free(__s);
35       key_txt = (FILE *)0x0;
36     }
37   }
38   return key_txt;
39 }
40
```

we can see that this code opens key.txt and reads its contents to a string __s then it compares this length with the length of **"THEY_WILL_NEVER_FIND_THIS"** which should be the same length. after that it creates another key s1 at 27 line.This key is being created by the decryption_key argument that if we see the previous function call it was the argument we pass with the wannabecry binary but with atoi :

```
encryption/decryption_KEY = atoi(*(char **)(param_2 + 8));
pFVar3 = FUN_00101bdc(encryption/decryption_KEY);
```

Now inside the CALL_CREATE_KEY function this happens:

```
1
2  void CALL_CREATE_KEY(char *param_1,int param_2)
3
4  {
5    CREATE_KEY(param_1,(95 - param_2 % 95) % 95);
6    return;
7  }
8
```

hmmmm interesting so we know that the argument we give with wannabecry to decrypt our files is doing this :

let's say our argument is called **arg** : (95 - atoi(**arg**) % 95) % 95 **(the + 8 is wrong)**
and then is passed to CREATE_KEY function which we analyzed as before and we hope that it returns **"THEY_WILL_NEVER_FIND_THIS"** so key_txt = 0 == 0 will return 1 and then in **figure 1** line 27 (1 == 0) resolve in false and go into **else** and decrypt our files

we basically need a number which given with the binary will reverse the key into its starting form "THEY_WILL_NEVER_FIND_THIS"
let's try to find that.

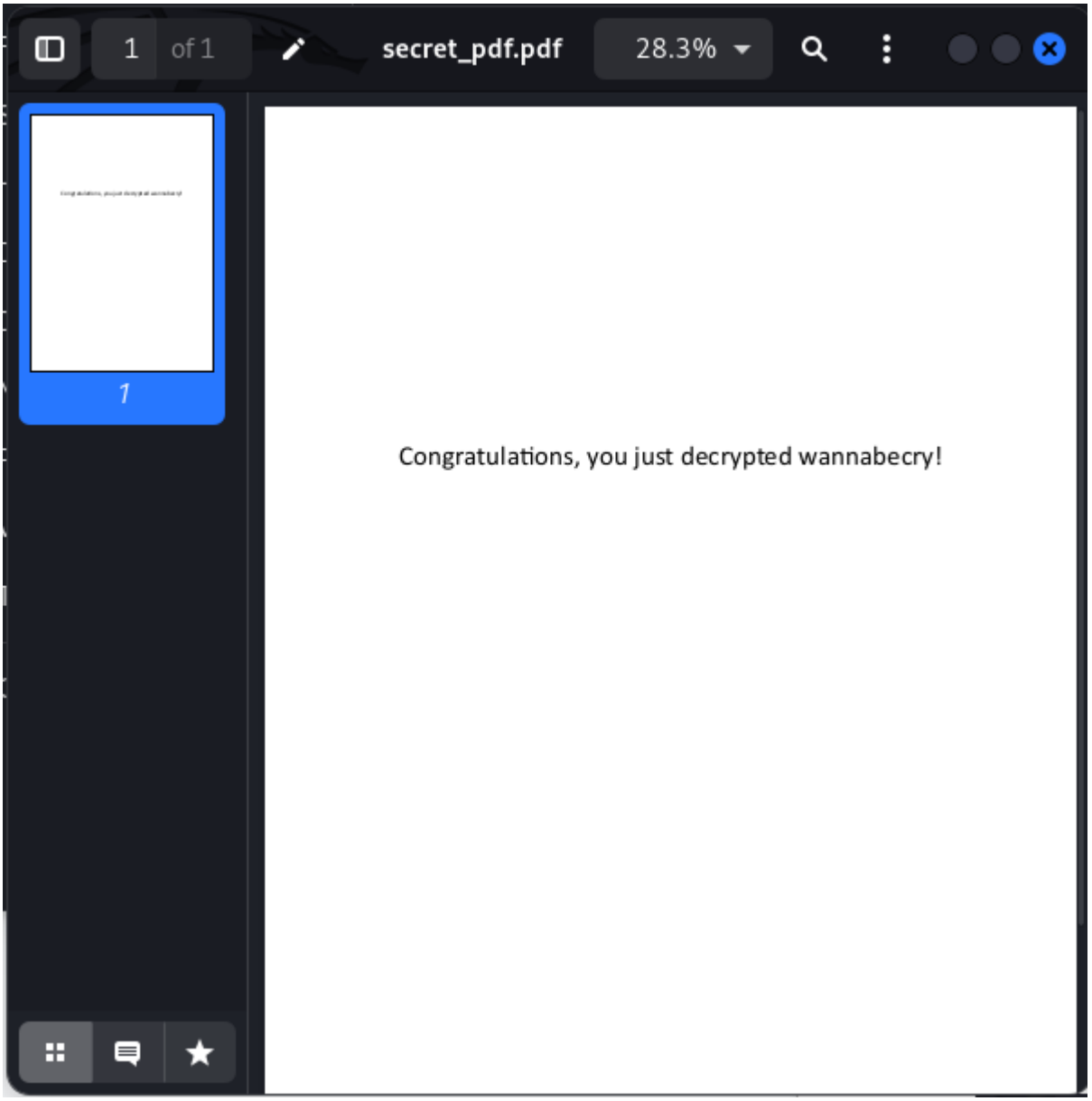
first lets find which number should CREATE_KEY take as the second arg so it reverses the param_1 (key.txt contents) in its first form lets see :
the first letter of the key is "**=**"
rember this ?

```
20    index_of_char_in_sequence = FIND_CHAR_NUM_FROM_0-94(ARG1[COUNT]);
21    if (index_of_char_in_sequence == -1) break;
22    ARG1_STR_MEM_ALOC[COUNT] =
23          " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuv
          wxyz{|}~"
24          [(param_2 + index_of_char_in_sequence) % 95];
```

"=" haves index = 31 in the sequence
and
"T" haves index = 54 in the sequence
so we need (param_2 + 31) % 95 = 54 solve this equation and we get **param_2 = 23**
okay lets try this as param_2 but first we need to solve:
(95 - atoi(**arg**) % 95) % 95 = 23 and we get **arg = 72** lets run this :

```
└─$ ./wannabecry 72
Decrypting files:
./virus_folder/bobs_secrets.txt
./virus_folder/secret_pdf.pdf
You did it! Your files are now decrypted!
```

WE GOT BOBS FILES BACK !


1 of 1
secret_pdf.pdf
28.3%
1
Congratulations, you just decrypted wannabecry!