GRANDMA'S PASSWORDS

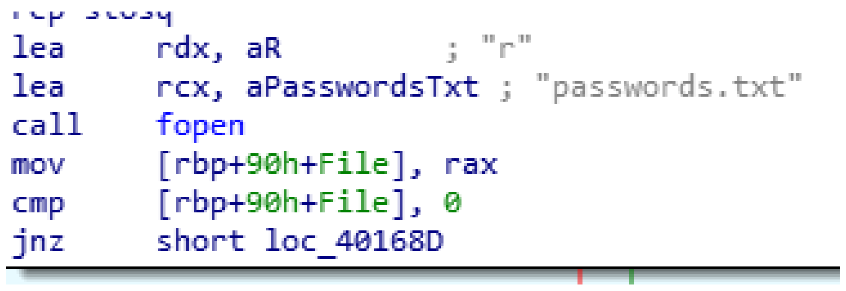
Description: My password file has been encrypted! Please help me get my passwords back. I can't log in to my WordsWithFriends account without it. :(

We're given two items: a passwords_encrypted.txt file and a ransom.exe file. If we try to run the ransom.exe, nothing will happen. Opening the passwords_encrypted.txt file gives us a bunch of garbage characters. Let's investigate the ransom.exe file.

Running the strings program doesn't give us any useful information, so let's take a look at it in IDA. (You can also use Ghidra if you prefer.)

This file came with debug information in it, which is really nice for us!

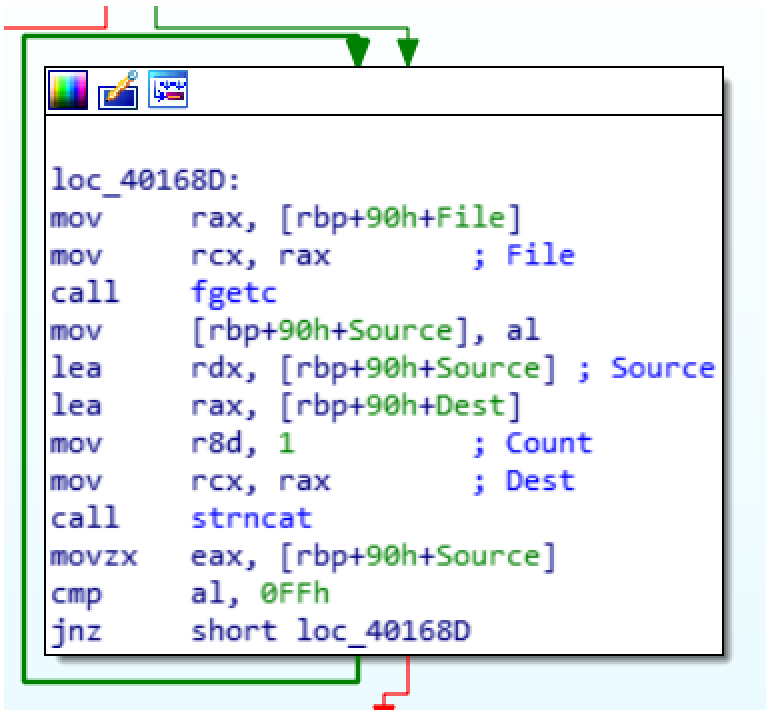
The first point of interest is this section at the start of the main function:



```
lea     rdx, aR          ; "r"
lea     rcx, aPasswordsTxt ; "passwords.txt"
call    fopen
mov     [rbp+90h+File], rax
cmp     [rbp+90h+File], 0
jnz     short loc_40168D
```

This appears to call the fopen function with two parameters: the string "passwords.txt" and "r". Since we know that the fopen function takes in a filename and a string representing the permissions, we can conclude that the program is looking for a file called "passwords.txt" and is opening it with read-only permissions. The program then checks to see if the file exists/opened correctly. If it didn't open correctly, the program returns a non-zero value and exits.

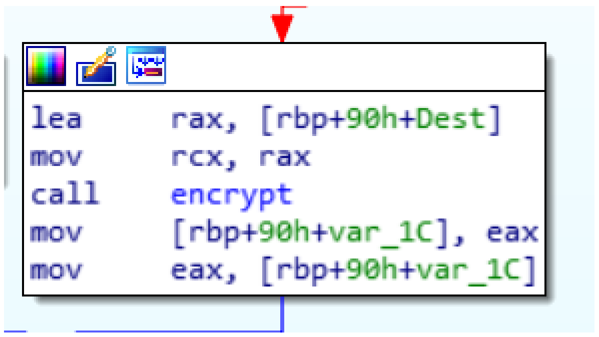
What's interesting is when the program finds the file.



```
loc_40168D:
mov     rax, [rbp+90h+File]
mov     rcx, rax          ; File
call    fgetc
mov     [rbp+90h+Source], al
lea     rdx, [rbp+90h+Source] ; Source
lea     rax, [rbp+90h+Dest]
mov     r8d, 1            ; Count
mov     rcx, rax          ; Dest
call    strncat
movzx   eax, [rbp+90h+Source]
cmp     al, 0FFh
jnz     short loc_40168D
```

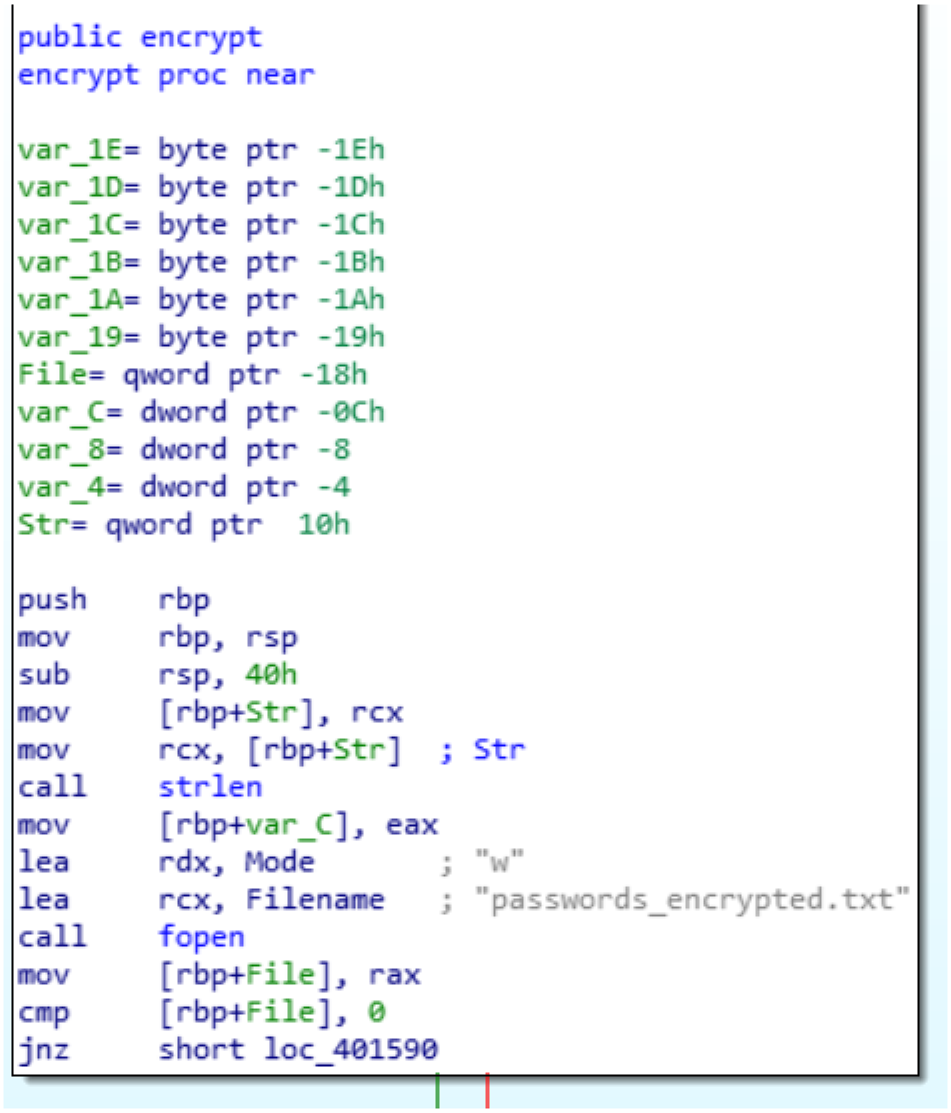
From the two function calls fgetc and strncat, we can see that this loops through the contents of the opened file and takes each character of the file and appends it to some string. This loops until an EOF (denoted by 0FFh), which upon reaching, the loop will break.

When the loop breaks, we get this block:



```
lea     rax, [rbp+90h+Dest]
mov     rcx, rax
call    encrypt
mov     [rbp+90h+var_1C], eax
mov     eax, [rbp+90h+var_1C]
```

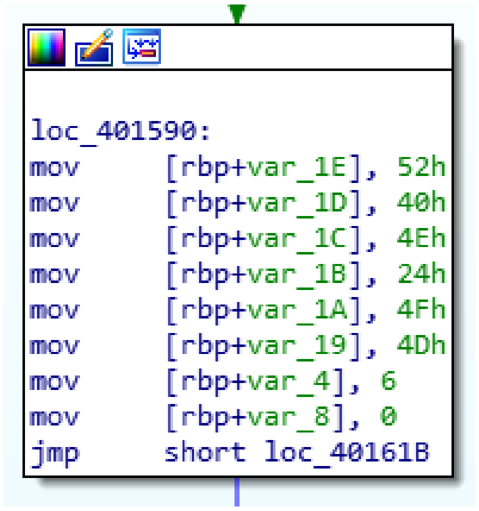
We know from the loop that the Dest variable is our concatenated string with the file contents. The program then calls an encrypt function. Let’s take a look at that.



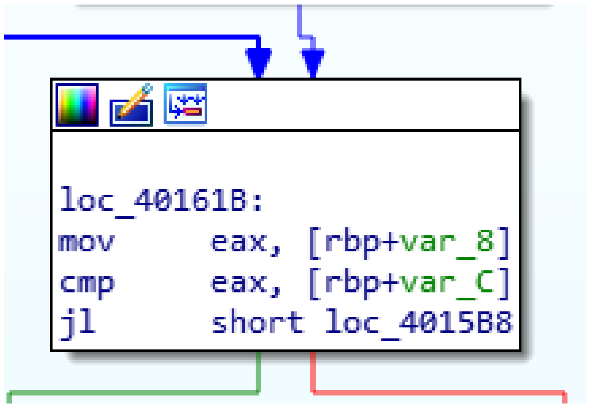
```
public encrypt
encrypt proc near

var_1E= byte ptr -1Eh
var_1D= byte ptr -1Dh
var_1C= byte ptr -1Ch
var_1B= byte ptr -1Bh
var_1A= byte ptr -1Ah
var_19= byte ptr -19h
File= qword ptr -18h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
Str= qword ptr  10h

push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     [rbp+Str], rcx
mov     rcx, [rbp+Str]  ; Str
call    strlen
mov     [rbp+var_C], eax
lea     rdx, Mode       ; "w"
lea     rcx, Filename   ; "passwords_encrypted.txt"
call    fopen
mov     [rbp+File], rax
cmp     [rbp+File], 0
jnz     short loc_401590
```

We are greeted with many local variables, most of which are unnamed, and an interesting set of instructions: we see the three fopen-related lines, similar to the ones we saw earlier. This time, we are passing a string “passwords_encrypted.txt” for the filename, and we are opening it with write permissions. We see the same check for validity afterwards with the same result.

If the file opens correctly, it enters another loop.

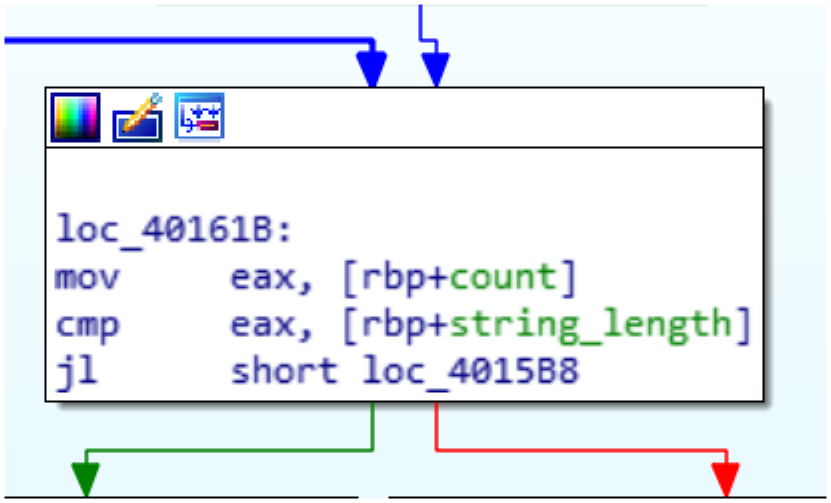

```
loc_401590:
mov     [rbp+var_1E], 52h
mov     [rbp+var_1D], 40h
mov     [rbp+var_1C], 4Eh
mov     [rbp+var_1B], 24h
mov     [rbp+var_1A], 4Fh
mov     [rbp+var_19], 4Dh
mov     [rbp+var_4], 6
mov     [rbp+var_8], 0
jmp     short loc_40161B
```

Before the loop, we see here that there are 6 hex values being moved into local variables, followed by a 6 and 0 being placed into other variables. Let's look into the loop to see how these are being used.


```
loc_40161B:
mov     eax, [rbp+var_8]
cmp     eax, [rbp+var_C]
jl      short loc_4015B8
```

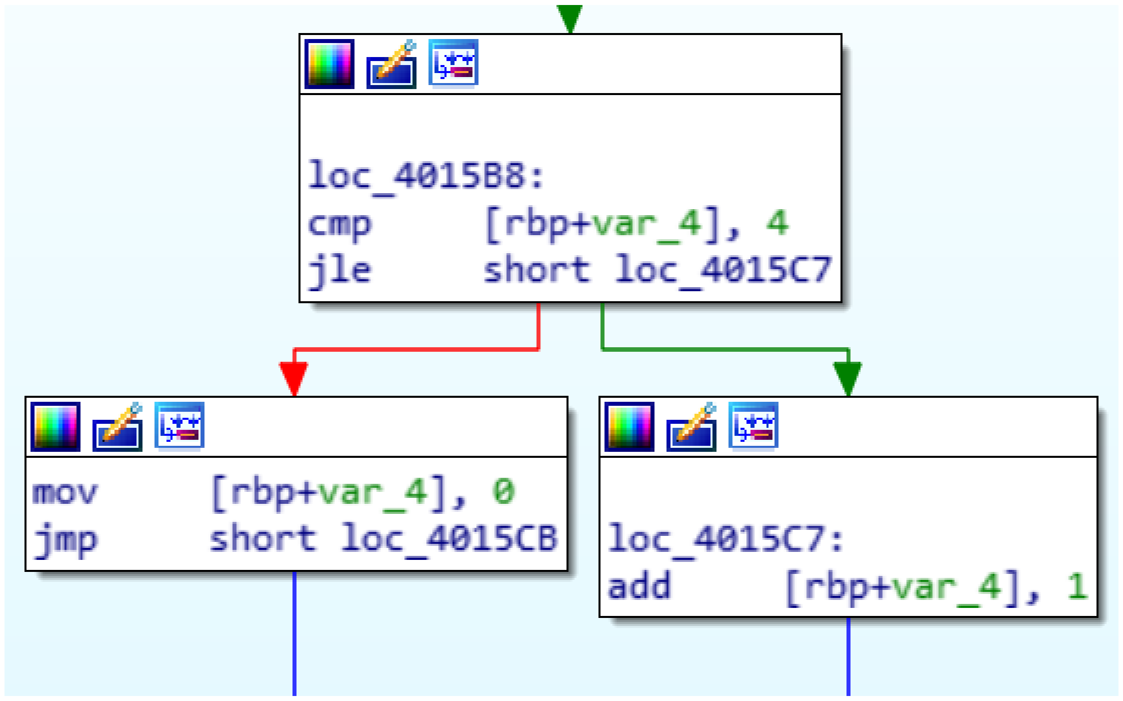
This here is our loop condition, or what determines whether or not there is another loop iteration. A variable (var_8) is moved into eax and compared with another variable (var_C). If var_8 is less than var_C, the loop exits.

var_8 was defined in our previous block as 0, and var_C was defined in the first block in this function as being the result of the strlen function. We can conclude from this that this block is checking to see if var_8 is less than the length of the string. We can rename both variables appropriately.



```
loc_40161B:
mov     eax, [rbp+count]
cmp     eax, [rbp+string_length]
jl      short loc_4015B8
```

The next set of blocks is a simple comparison.

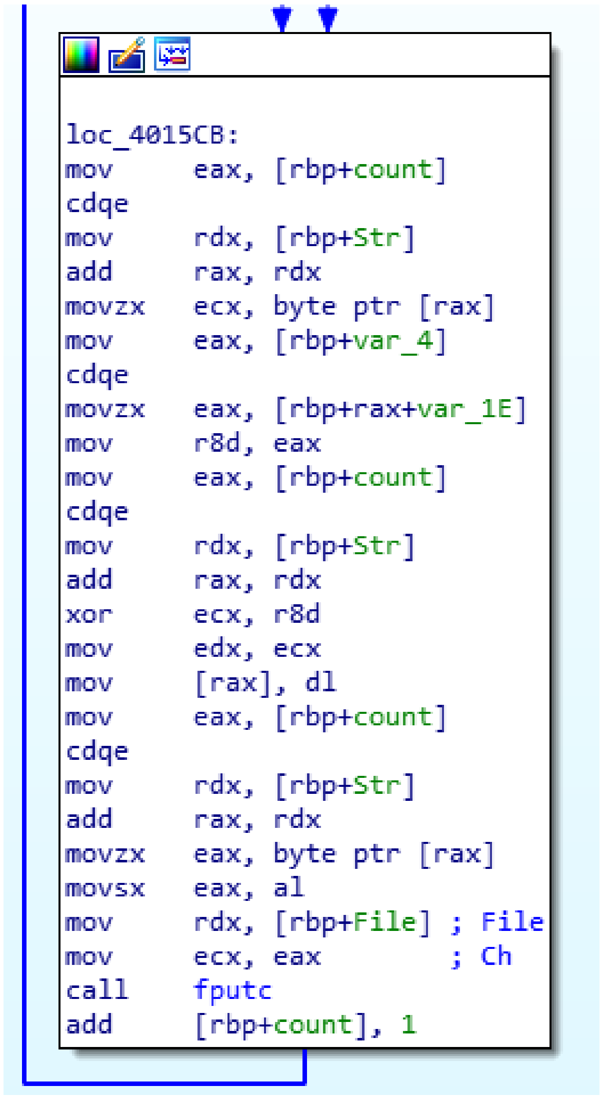


```
loc_4015B8:
cmp     [rbp+var_4], 4
jle     short loc_4015C7
```

```
mov     [rbp+var_4], 0
jmp     short loc_4015CB
```

```
loc_4015C7:
add     [rbp+var_4], 1
```

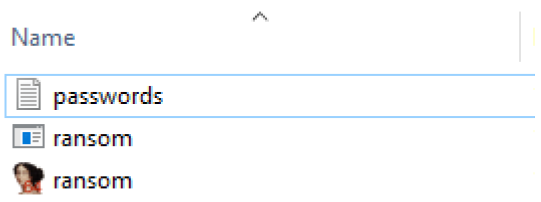
We see that the value in var_4 is being compared to the number 4. If that value is less than or equal to 4, var_4 gets incremented by 1. Otherwise, var_4 gets reset to 0. Regardless, both blocks point to the same place after each condition.



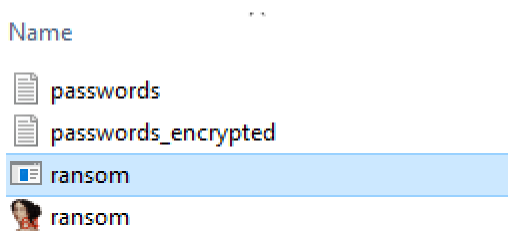
```
loc_4015CB:
mov     eax, [rbp+count]
cdqe
mov     rdx, [rbp+Str]
add     rax, rdx
movzx   ecx, byte ptr [rax]
mov     eax, [rbp+var_4]
cdqe
movzx   eax, [rbp+rax+var_1E]
mov     r8d, eax
mov     eax, [rbp+count]
cdqe
mov     rdx, [rbp+Str]
add     rax, rdx
xor     ecx, r8d
mov     edx, ecx
mov     [rax], dl
mov     eax, [rbp+count]
cdqe
mov     rdx, [rbp+Str]
add     rax, rdx
movzx   eax, byte ptr [rax]
movsx   eax, al
mov     rdx, [rbp+File] ; File
mov     ecx, eax        ; Ch
call    fputc
add     [rbp+count], 1
```

This block looks daunting, but there isn't a whole lot going on in here. The first half (before the XOR instruction) is finding a specific character in the string and an offset after the var_1E variable and XORing them. After the XOR, the result of that instruction gets written to the file that was created at the start of the function.

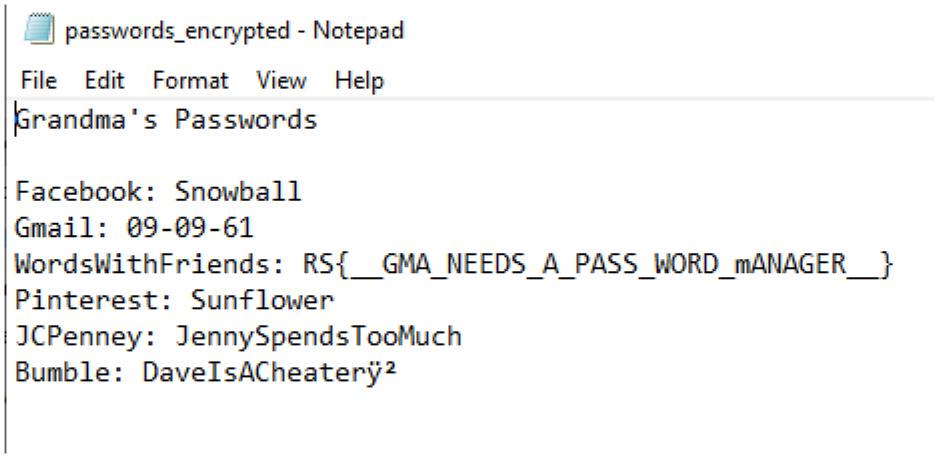
From this, we can determine that the contents of the passwords.txt file are being read into a string, which is then iterated through a loop. Each character of the string is XORed with one of the hex values placed into the local variables. Therefore, we can conclude that the contents of the passwords.txt file are being encrypted over a 6-character key (represented by the hex values). Since XOR is reversible, if we rename the passwords_encrypted.txt file to passwords.txt and run the program, the program will decrypt the contents for us. The result will be in a new file named passwords_encrypted.txt.



Rename the file to “passwords.txt”. Then, run the ransom.exe program.



A “passwords_encrypted.txt” file will be created.



```
Grandma's Passwords

Facebook: Snowball
Gmail: 09-09-61
WordsWithFriends: RS{__GMA_NEEDS_A_PASS_WORD_mANAGER__}
Pinterest: Sunflower
JCPenney: JennySpendsTooMuch
Bumble: DaveIsACheaterÿ²
```

You can open the file and find the restored contents as well as the flag.