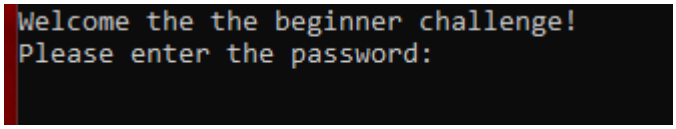
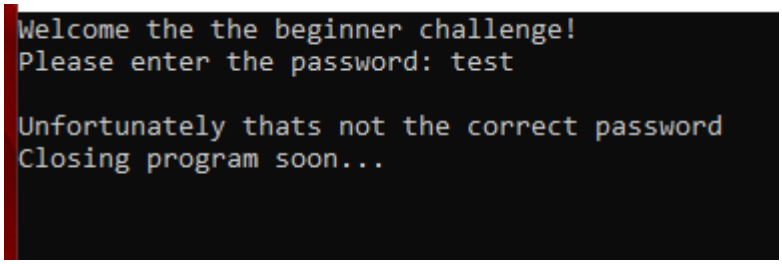
Basic Challenge 2

S Chowdhury

Walkthrough for basic challenge 2. Run the program.

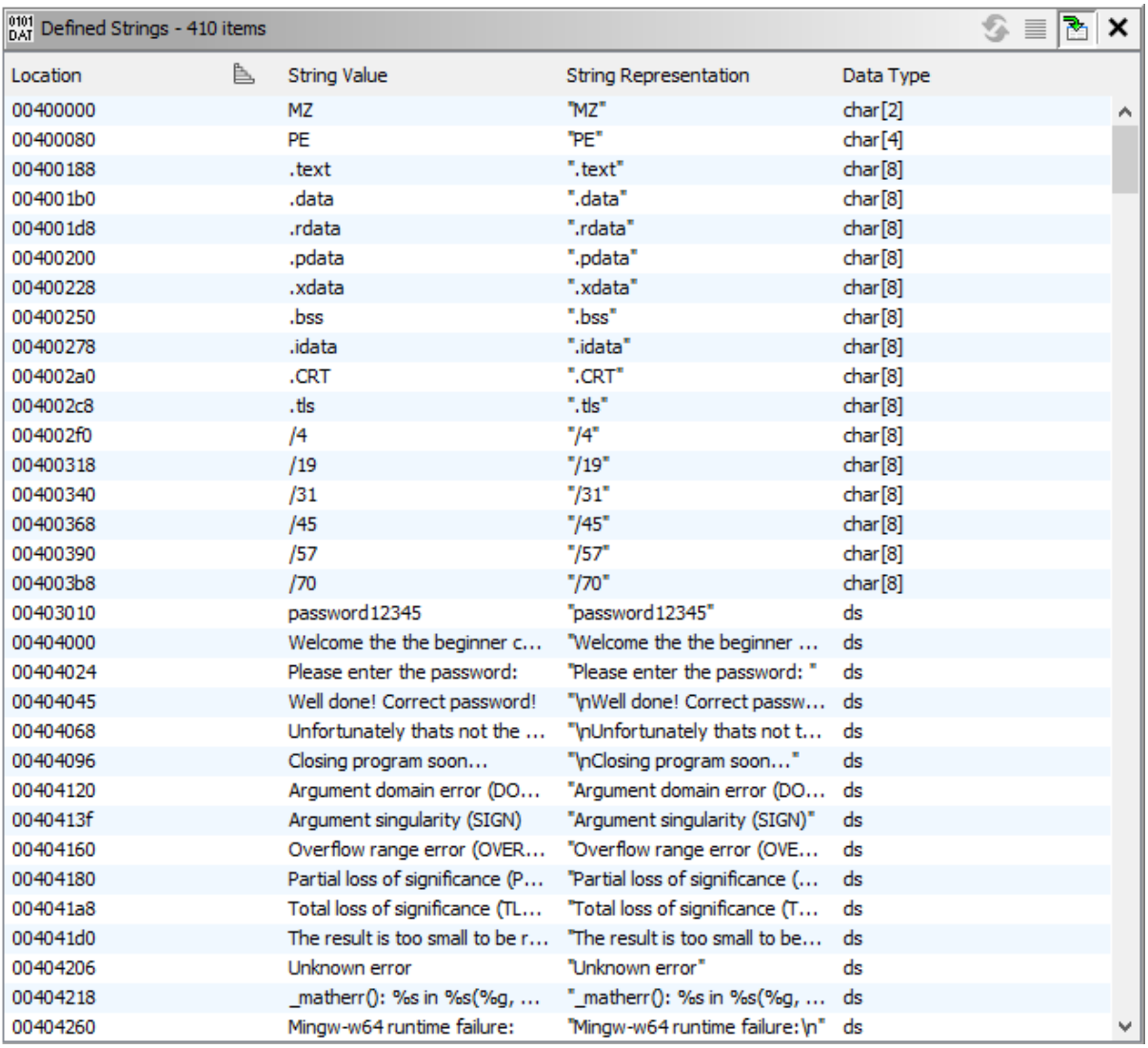


Try “test” for password.



Open this program in Ghidra so we can investigate.

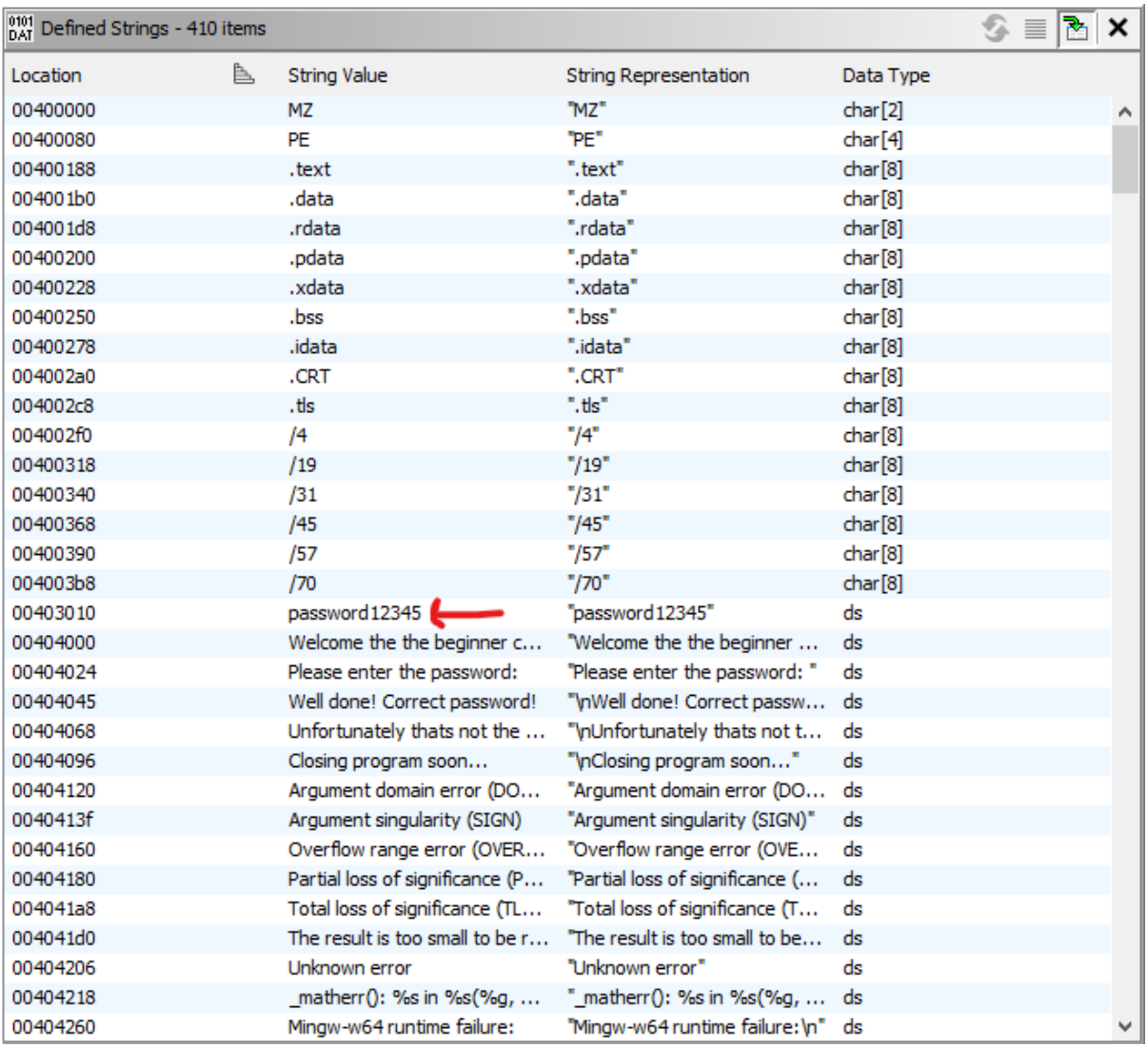
Select *Window>Defined Strings* to check the strings.



Defined Strings - 410 items

Location	String Value	String Representation	Data Type
00400000	MZ	"MZ"	char[2]
00400080	PE	"PE"	char[4]
00400188	.text	".text"	char[8]
004001b0	.data	".data"	char[8]
004001d8	.rdata	".rdata"	char[8]
00400200	.pdata	".pdata"	char[8]
00400228	.xdata	".xdata"	char[8]
00400250	.bss	".bss"	char[8]
00400278	.idata	".idata"	char[8]
004002a0	.CRT	".CRT"	char[8]
004002c8	.tls	".tls"	char[8]
004002f0	/4	"/4"	char[8]
00400318	/19	"/19"	char[8]
00400340	/31	"/31"	char[8]
00400368	/45	"/45"	char[8]
00400390	/57	"/57"	char[8]
004003b8	/70	"/70"	char[8]
00403010	password12345	"password12345"	ds
00404000	Welcome the the beginner c...	"Welcome the the beginner ...	ds
00404024	Please enter the password:	"Please enter the password: "	ds
00404045	Well done! Correct password!	"\nWell done! Correct passw...	ds
00404068	Unfortunately thats not the ...	"\nUnfortunately thats not t...	ds
00404096	Closing program soon...	"\nClosing program soon..."	ds
00404120	Argument domain error (DO...	"Argument domain error (DO...	ds
0040413f	Argument singularity (SIGN)	"Argument singularity (SIGN)"	ds
00404160	Overflow range error (OVER...	"Overflow range error (OVE...	ds
00404180	Partial loss of significance (P...	"Partial loss of significance (...	ds
004041a8	Total loss of significance (TL...	"Total loss of significance (T...	ds
004041d0	The result is too small to be r...	"The result is too small to be...	ds
00404206	Unknown error	"Unknown error"	ds
00404218	_matherr(): %s in %s(%g, ...	"_matherr(): %s in %s(%g, ...	ds
00404260	Mingw-w64 runtime failure:	"Mingw-w64 runtime failure:\n"	ds

We can see the “password12345” just like the previous challenge.



Defined Strings - 410 items

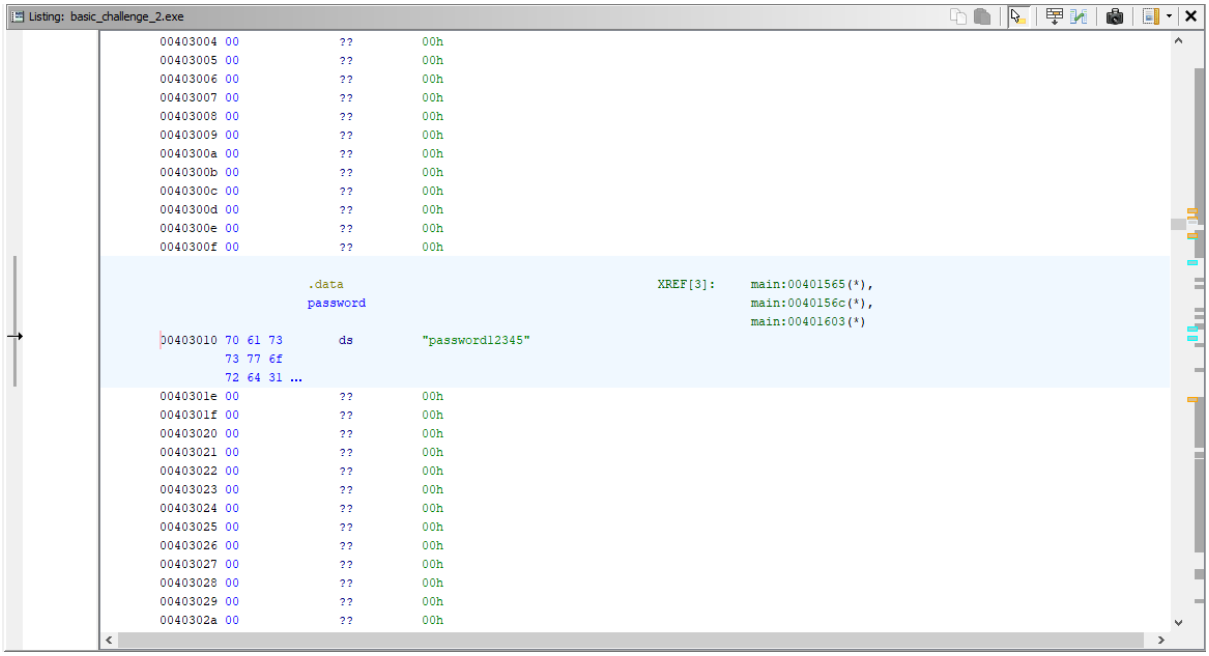
Location	String Value	String Representation	Data Type
00400000	MZ	"MZ"	char[2]
00400080	PE	"PE"	char[4]
00400188	.text	".text"	char[8]
004001b0	.data	".data"	char[8]
004001d8	.rdata	".rdata"	char[8]
00400200	.pdata	".pdata"	char[8]
00400228	.xdata	".xdata"	char[8]
00400250	.bss	".bss"	char[8]
00400278	.idata	".idata"	char[8]
004002a0	.CRT	".CRT"	char[8]
004002c8	.tls	".tls"	char[8]
004002f0	/4	"/4"	char[8]
00400318	/19	"/19"	char[8]
00400340	/31	"/31"	char[8]
00400368	/45	"/45"	char[8]
00400390	/57	"/57"	char[8]
004003b8	/70	"/70"	char[8]
00403010	password12345	"password12345"	ds
00404000	Welcome the the beginner c...	"Welcome the the beginner ...	ds
00404024	Please enter the password:	"Please enter the password: "	ds
00404045	Well done! Correct password!	"\nWell done! Correct passw...	ds
00404068	Unfortunately thats not the ...	"\nUnfortunately thats not t...	ds
00404096	Closing program soon...	"\nClosing program soon..."	ds
00404120	Argument domain error (DO...	"Argument domain error (DO...	ds
0040413f	Argument singularity (SIGN)	"Argument singularity (SIGN)"	ds
00404160	Overflow range error (OVER...	"Overflow range error (OVE...	ds
00404180	Partial loss of significance (P...	"Partial loss of significance (...	ds
004041a8	Total loss of significance (TL...	"Total loss of significance (T...	ds
004041d0	The result is too small to be r...	"The result is too small to be...	ds
00404206	Unknown error	"Unknown error"	ds
00404218	_matherr(): %s in %s(%g, ...	"_matherr(): %s in %s(%g, ...	ds
00404260	Mingw-w64 runtime failure:	"Mingw-w64 runtime failure:\n"	ds

Try entering this as the password:

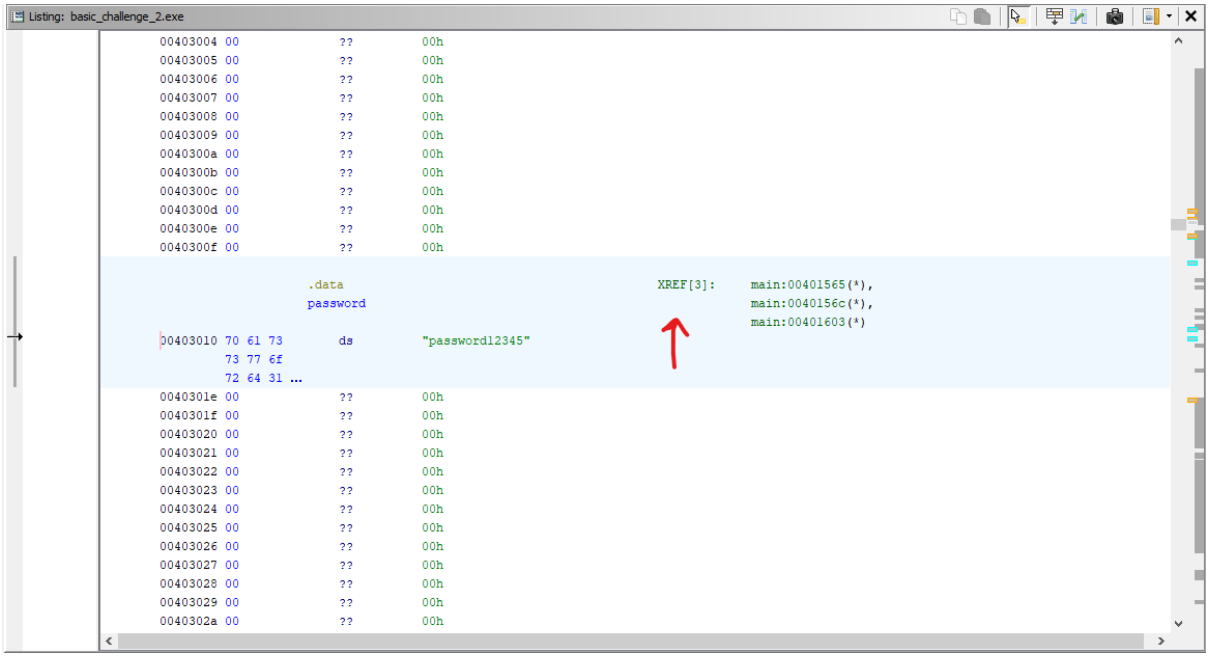
```
Welcome the the beginner challenge!
Please enter the password: password12345

Unfortunately thats not the correct password
Closing program soon...
```

Still incorrect.

We can double click on this “password12345” in Ghidra to see this data in the **Listing Window**.



Notice the **XREF** on the right side:



We can see three memory addresses next to the **XREF**:

- 00401565
- 0040156c
- 00401603

At these memory addresses, this global variable gets referenced. Lets look at those addresses, you can just click on the address on the **XREF**, or select *Navigation>Go To* option then enter memory address. Here's what we can see at these addresses:

00401565

```
          00 00
0040155d 48 8d 45 c0     LEA         RAX,[RBP + -0x40]
00401561 48 89 45 f0     MOV         qword ptr [RBP + -0x10],RAX
00401565 48 8d 05        LEA         RAX,[password]
         a4 1a 00 00
```

0040156c

```
0040156c 48 89 45 e8     MOV         qword ptr [RBP + -0x18],RAX=>password          = "password12345"
00401570 48 8d 0d        LEA         RCX,[s_Welcome_the_the_beginner_challen_004040... = "Welcome the the beginner chal...
         89 2a 00 00
00401577 e8 f4 15        CALL        puts                                           undefined puts()
         00 00
0040157c 48 8d 0d        LEA         RCX,[s_Please_enter_the_password:_00404024]     = "Please enter the password: "
         a1 2a 00 00
00401583 e8 f0 15        CALL        printf                                         undefined printf()
         00 00
```

00401603

```
00401603 48 8b 55 e8     MOV         RDX=>password,qword ptr [RBP + -0x18]          = "password12345"
00401607 48 8b 45 f0     MOV         RAX,qword ptr [RBP + -0x10]
0040160b 48 89 c1        MOV         RCX,RAX
0040160e e8 45 15        CALL        strcmp                                         undefined strcmp()
         00 00
```

Lets open the decompiler at the *main* function. Click *Window>Decompiler* if it's not open.

```
1
2 int __cdecl main(int _Argc,char **_Argv,char **_Env)
3
4 {
5   size_t sVar1;
6   char local_48 [28];
7   int local_2c;
8   char local_25;
9   int local_24;
10  char *local_20;
11  char *local_18;
12  int local_c;
13
14  __main();
15  local_18 = local_48;
16  local_20 = password;
17  puts("Welcome the the beginner challenge!");
18  printf("Please enter the password: ");
19  scanf("%19s",local_48);
20  sVar1 = strlen(local_48);
21  local_24 = (int)sVar1;
22  for (local_c = 0; local_c < local_24 / 2; local_c = local_c + 1) {
23    local_25 = local_48[local_c];
24    local_48[local_c] = local_48[(local_24 + -1) - local_c];
25    local_48[(local_24 + -1) - local_c] = local_25;
26  }
27  local_2c = strcmp(local_18,local_20);
28  if (local_2c == 0) {
29    printf("\nWell done! Correct password!");
30  }
31  else {
32    printf("\nUnfortunately thats not the correct password");
```

From looking at this code, here's information we can figure out:

- On line 6, local_48 seems to be the buffer. On line 19, we can see that this variable gets put into the *scanf* function in line 19, so the user input gets put here. We can rename this variable to "buffer".
- From line 11, we can see local_18 seems to be a pointer. From line 15, we can see it's a pointer to local_48, which we know is the buffer. Let's call local_18 as "buffer_ptr".
- From line 10, we can see that local_20 seems to be a char pointer. From line 16, we can see this points to the password global variable. Rename this pointer to "password_ptr".
- On line 22, we can see that local_c probably the looping index. Rename this to "ii".
- On line 19 the user input gets saved into local_48 and on line 20, we check the length of this string with the *strlen* function. This length gets stored into sVar1. We can rename sVar1 as "input_length".
- On line 27, local_2c stores the result of *strcmp*. Rename as "result".
- On line 21, local_21 seems to be the same as sVar1, rename as "input_length_int".

```
1
2 int __cdecl main(int _Argc,char **_Argv,char **_Env)
3
4 {
5   size_t input_length;
6   char buffer [28];
7   int result;
8   char local_25;
9   int input_length_int;
10  char *password_ptr;
11  char *buffer_ptr;
12  int ii;
13
14  __main();
15  buffer_ptr = buffer;
16  password_ptr = password;
17  puts("Welcome the the beginner challenge!");
18  printf("Please enter the password: ");
19  scanf("%19s",buffer);
20  input_length = strlen(buffer);
21  input_length_int = (int)input_length;
22  for (ii = 0; ii < input_length_int / 2; ii = ii + 1) {
23    local_25 = buffer[ii];
24    buffer[ii] = buffer[(input_length_int + -1) - ii];
25    buffer[(input_length_int + -1) - ii] = local_25;
26  }
27  result = strcmp(buffer_ptr,password_ptr);
28  if (result == 0) {
29    printf("\nWell done! Correct password!");
30  }
31  else {
32    printf("\nUnfortunately thats not the correct password");
```

From line 22 to line 26, we can see that the buffer array gets manipulated somehow. For *ii=0*, while *ii* less than *input_length_int/2*, with *ii* getting incremented by *1* each iteration of the loop. From line 23, 24, and 25 we can see that the value at *buffer[ii]* gets swapped with the value at *buffer[input_length_int -ii -1]*. Since this is happening for *ii=0* until it reaches *input_length_int/2*, this code reverses the string. After that, the *strcmp* function gets called.

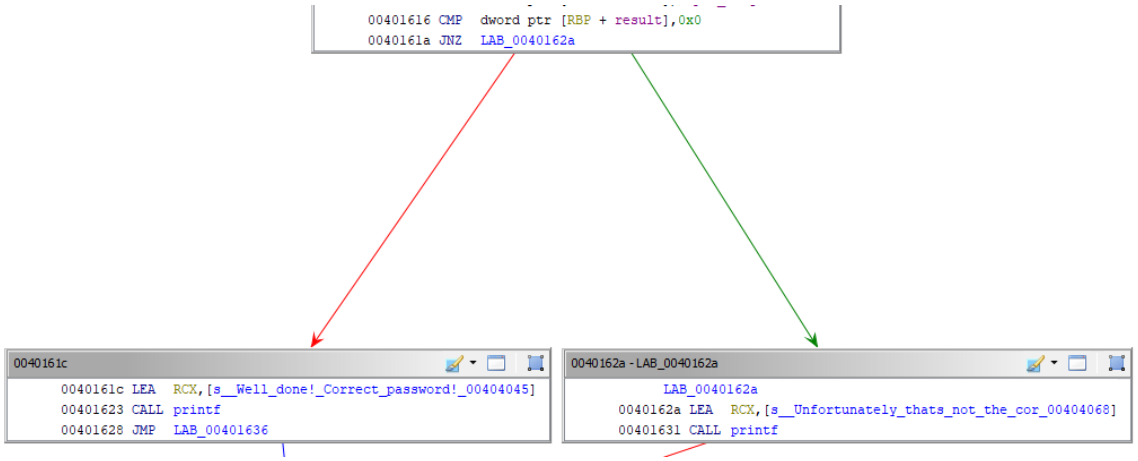
Lets try inputting the reverse of “password12345” as the password:

```
Welcome the the beginner challenge!
Please enter the password: 54321drowssap

Well done! Correct password!
Closing program soon...
```

Bonus Section

From Ghidra, we can patch instructions. Open the function graph for main:

Scroll down to where the *cmp* instruction gets performed right before printing out the well done or unfortune message:

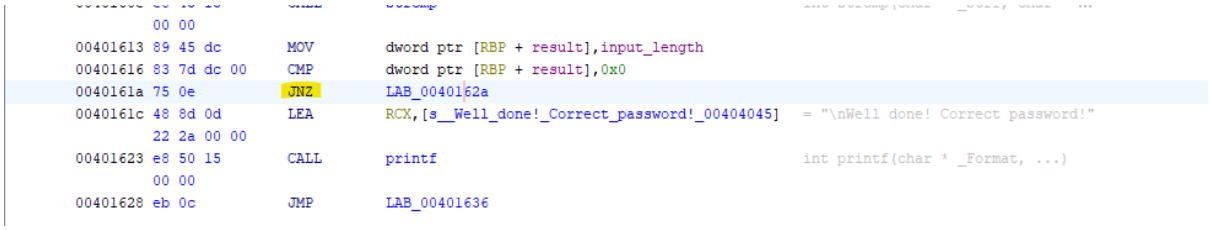


At **0040161a**, the *JNZ* instruction will perform a jump to the unfortunate message based on whether or not the zero flag gets set. In this case it would jump if the 0 flag is not set.

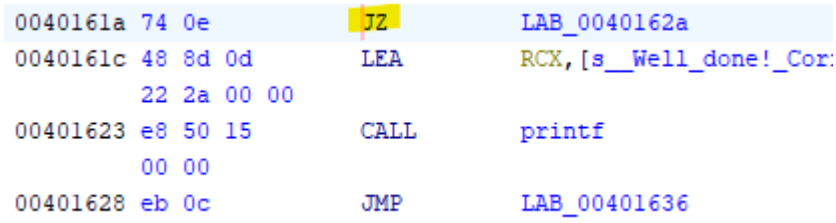
At **00401616**, the *cmp* function compares the result of the *strcmp* function with 0x0, and if the result equals to 0, the zero flag in the flags register gets set.

However, what if we change the *JNZ* instruction to a *JZ* instruction? Then we would jump when the zero flag gets set. In other words, it would jump to the unfortunate message if we enter the correct password, and enter the success message when we enter the wrong password.

Return back to normal view:

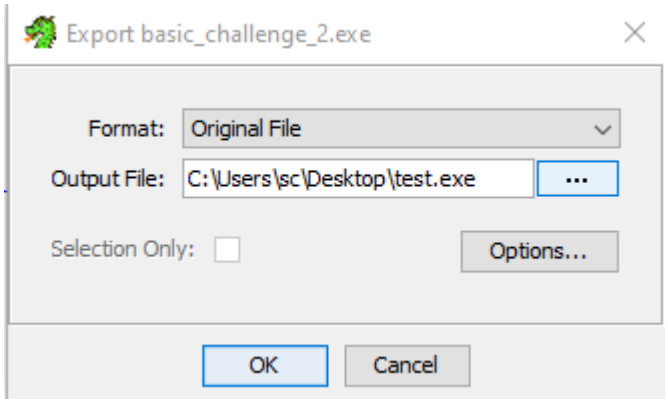


Right click *JNZ* then select *Patch Instruction*. Then change the *JNZ* to *JZ*. When the *74 0e* option comes up, click that.

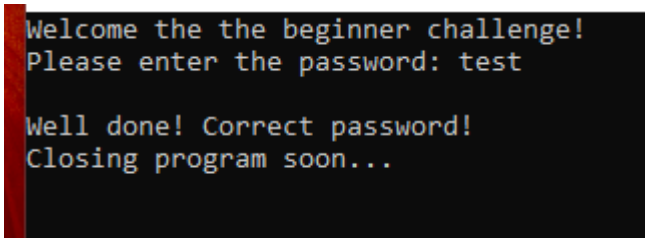


We can export this new patched version by clicking *File>Export Program*.

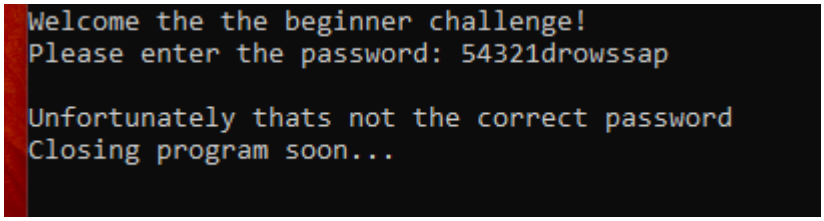
Set format as “original file” then select where to export.



Now lets run this program and enter “test”:



It says correct password. What if we put “54321drowssap”?



It says wrong password. This is because we changed the *JNZ* to *JZ*.