Freemasonry

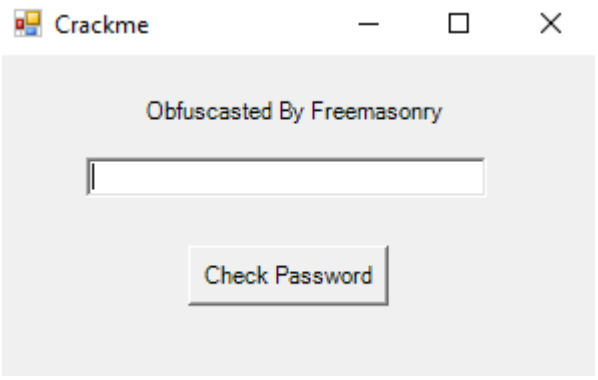
Capture the Flag Challenge.

Link: Challenge can be found here.

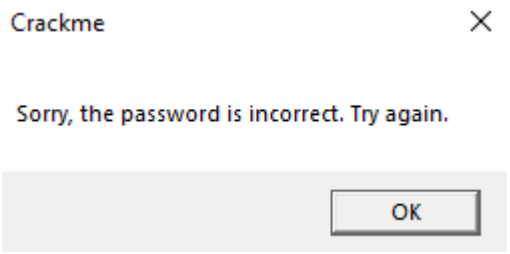
Overview: The objective of this CTF is to detect the correct password to crack the program.

The program is 32bit .NET exe.

Upon running the program (on virtual machine, for safety reasons), the user is requested to enter password:



For entering custom password, we get the message:



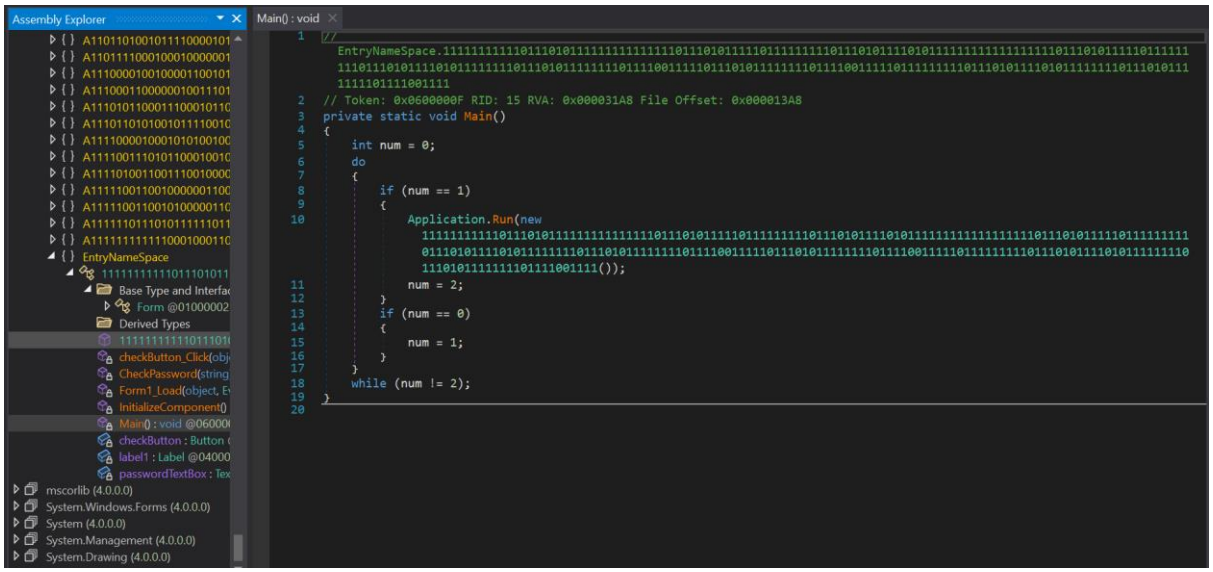
Method:

As we are dealing with 32bit .NET exe, the first order of business is to decompile the program with dnSpy:

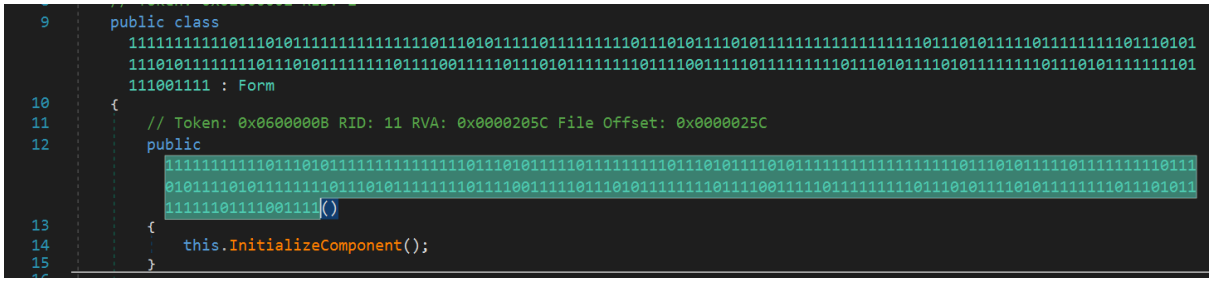
The first thing that was observed that they were kind enough to provide the namespace which is used as entry point:

```
// Entry point:
EntryNameSpace.1111111111101110101111111111111111011101011111011111111101110101111010111111111111111111011101011111011111
11101110101111010111111111011101010111111110111100111110111010111111111011110011111011111111111011101011110101111111101110101111
1111101111001111.Main
// Timestamp: <Unknown> (FD8507A4)
```

I renamed that namespace for easy identification and opened the main function in it:

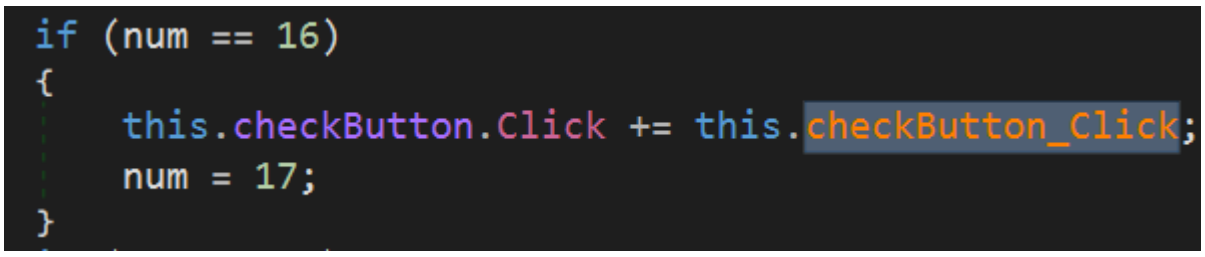


It runs an object of class of the same string they gave us, I took a look in it:



It runs the ‘InitializeComponent’ method.

The relevant part in the ‘InitializeComponent’ method is that when the user clicks on the ‘check password’ button -it runs checkButton_Click function



Let’s examine ‘checkButton_Click’:

```
335         // Token: 0x0600000D RID: 13 RVA: 0x00002F2C File Offset: 0x0000112C
336         private void checkButton_Click(object sender, EventArgs e)
337         {
338             int num = 0;
339             for (;;)
340             {
341                 string text;
342                 if (num == 1)
343                 {
344                     text = this.passwordTextBox.Text;
345                     num = 2;
346                 }
347                 bool flag;
348                 if (num == 2)
349                 {
350                     flag = this.CheckPassword(text);
351                     num = 3;
352                 }
353                 if (num == 4)
354                 {
355                     MessageBox.Show(Encoding.UTF8.GetString(Convert.FromBase64String
                          ("Q29uZ3JhdHVsYXRpb25zISBZb3UgaGF2ZSBzdWNjZXNzZnVsbHkgY3JhY2tlZCB0aGUgcGFzc3dvcmQu")),
                          Encoding.UTF8.GetString(Convert.FromBase64String("Q3JhY2ttZQ==")));
356                     num = 5;
357                 }
358                 if (num == 5)
359                 {
360                     goto IL_12E;
361                 }
362                 if (num == 6)
363                 {
364                     goto IL_E2;
365                 }
366                 goto IL_11C;
367                 IL_140:
368                 if (num == 3)
369                 {
370                     if (!flag)
371                     {
372                         goto IL_E2;
373                     }
374                     num = 4;
375                 }
376                 if (num == 0)
377                 {
378                     num = 1;
379                 }
```

It can be observed that the function has variable ‘num’ which its initial value is 0. Then a loop is being run and on the first iteration the ‘num’ value is being set to 1. In the next iteration a variable ‘text’ gets the value of the user input string, and ‘num’ is being set to 2.

On the next iteration the ‘text’ variable is being inserted to ‘checkPassword’ method, lets take a look in it:

```
402         private bool CheckPassword(string password)
403         {
404             int num = 0;
405             bool flag;
406             do
407             {
408                 string @string;
409                 if (num == 1)
410                 {
411                     @string = Encoding.UTF8.GetString(Convert.FromBase64String
                          ("MDEwMDAxMTAwMTAxMDAxMDAxMDAwMTAxMDEwMDAxMDEwMTAwMTEwMTAxMDAwMDAxMDEwMTAwMTEwMTAwMTExMTAxMDAxMTEwMDE
                          wMTAwMTAwMTAxMTAwMQ=="));
412                     num = 2;
413                 }
414                 if (num == 2)
415                 {
416                     flag = password.Equals(@string);
417                     num = 3;
418                 }
419                 if (num == 3)
420                 {
421                     break;
422                 }
423                 if (num == 0)
424                 {
425                     num = 1;
426                 }
427             }
428             while (num != 4);
429             return flag;
430         }
431
```

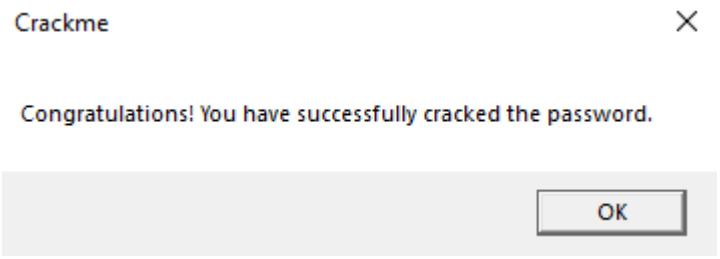
This function also has ‘num’ variable initialized with the value 0, then being set to 1. And then there is a variable ‘@string’ is being checked to the base64 decoding of some string, and that decoded value is being compared to the password.

It means – that base64 decoded string is the password.

The base64 decoding of the displayed string is:

01000110010100100100010101000101010011010100000101010011010011110100111001010010010011001

Let’s check it:



Success, the password for this program was found!

Conclusions: This challenge was cute and easy. Nothing too difficult, but it was useful in order to deepening existing experience in reverse engineering .NET applications, for upcoming, more challenging Capture the flags.