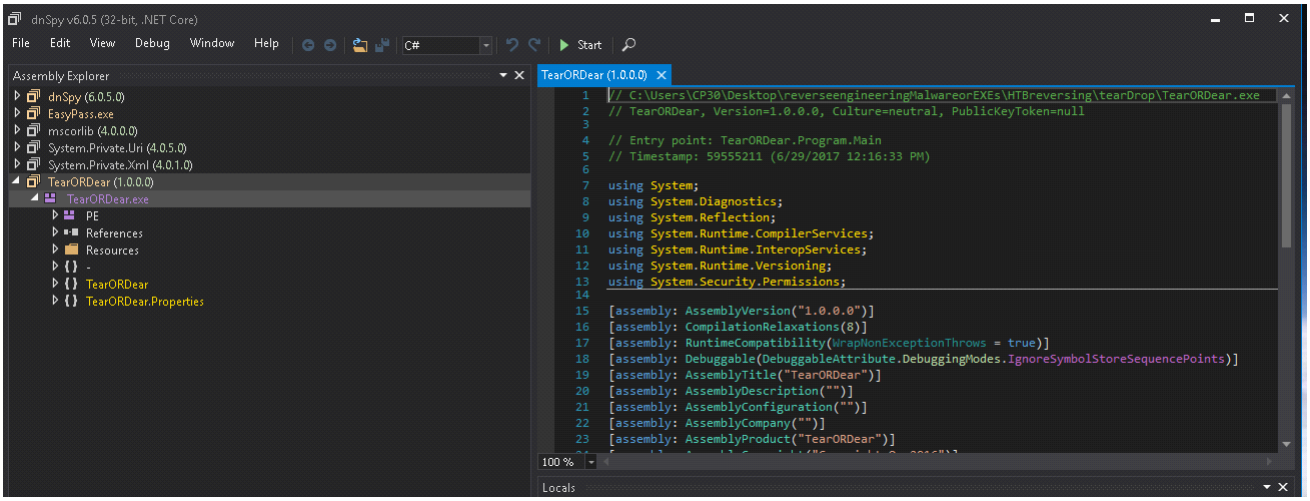
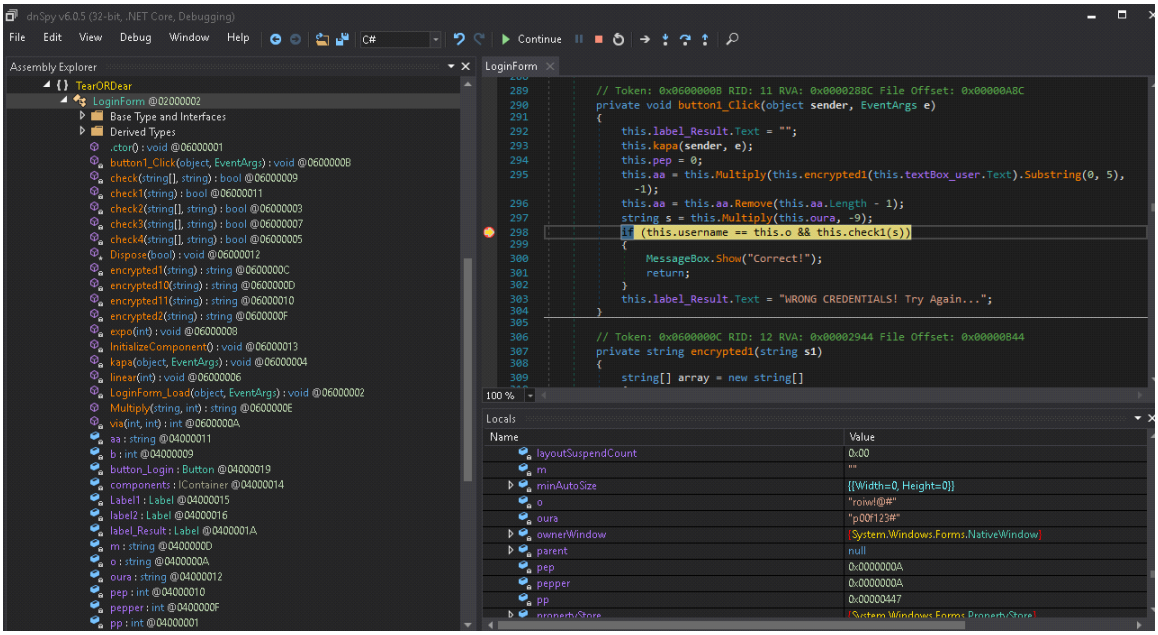
TearOrDear.exe

Hack The Box Challenge



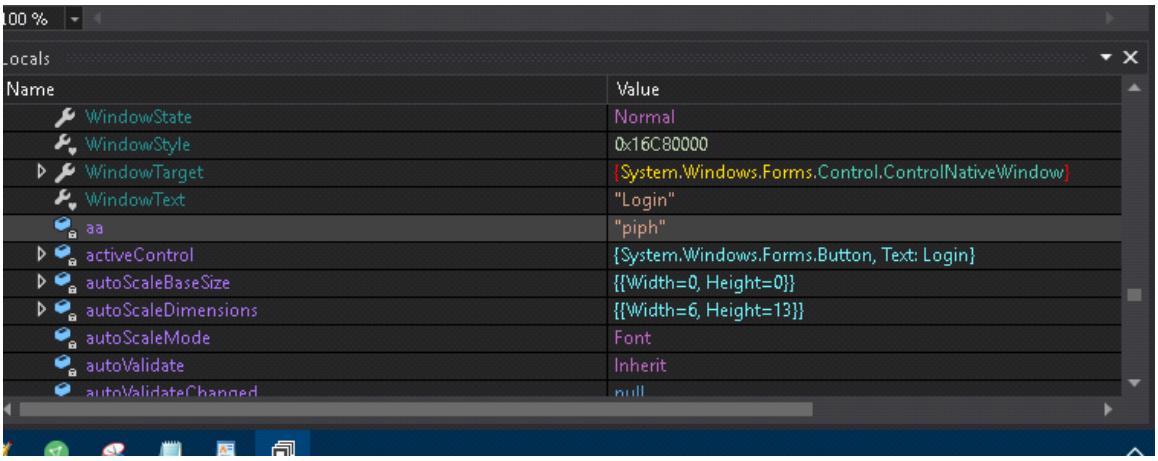
By using a nice tool called DnSpy available on GitHub https://github.com/0xd4d/dnSpyHere we now are able to produce the source code.

Getting started with tear or dear an Hack the Box challenge. By using PEid I was able to determine right away it was a .Net 32 bit executable. A trick all learning reverse engineering there is always in the corner of the hex output a MZ in the corner indicates a microsoft binary. It will as well reveal the values needed including labeling it a .Net application.

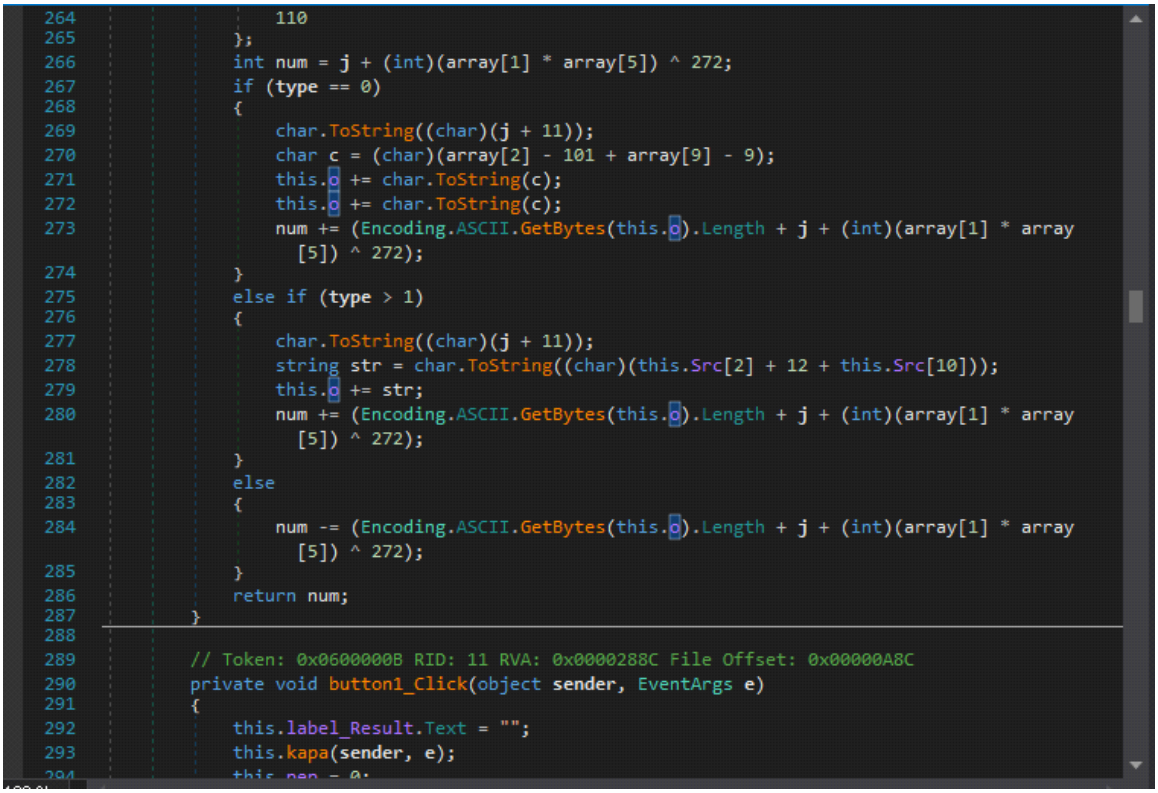


With this one the easy way to get were we need to go since we got raw source code decompiled is to see some values of the variables in the code. So to get this done we set a

break point on line 298. We set one here as we can see by running it the dialog box pops up we give it some values and it stops at our breakpoint now.



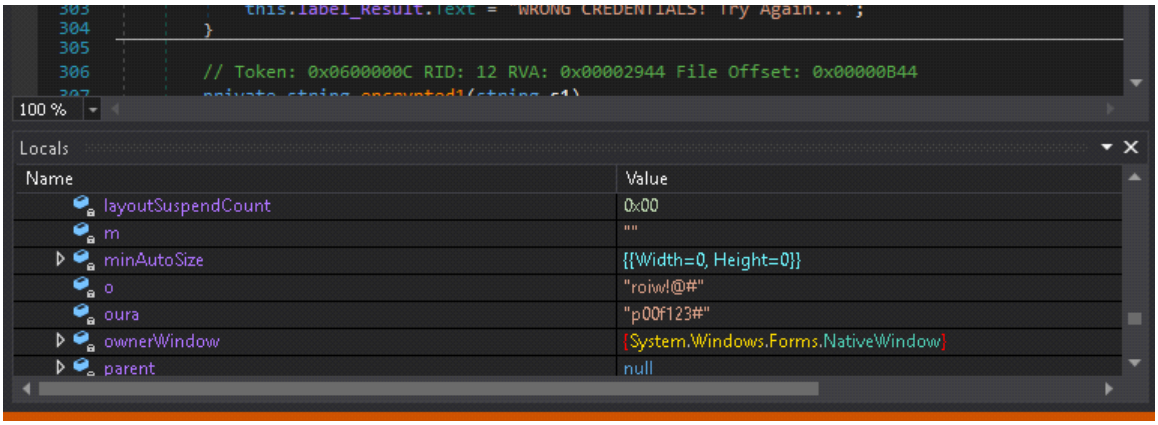
Digging for our first variable we find the value now of aa and reads "piph" which should in theory by reading our code the username. Here this has many rabbit holes so by setting our breakpoint we can dive in see exactly what values are being compared. Now we start digging for our password.



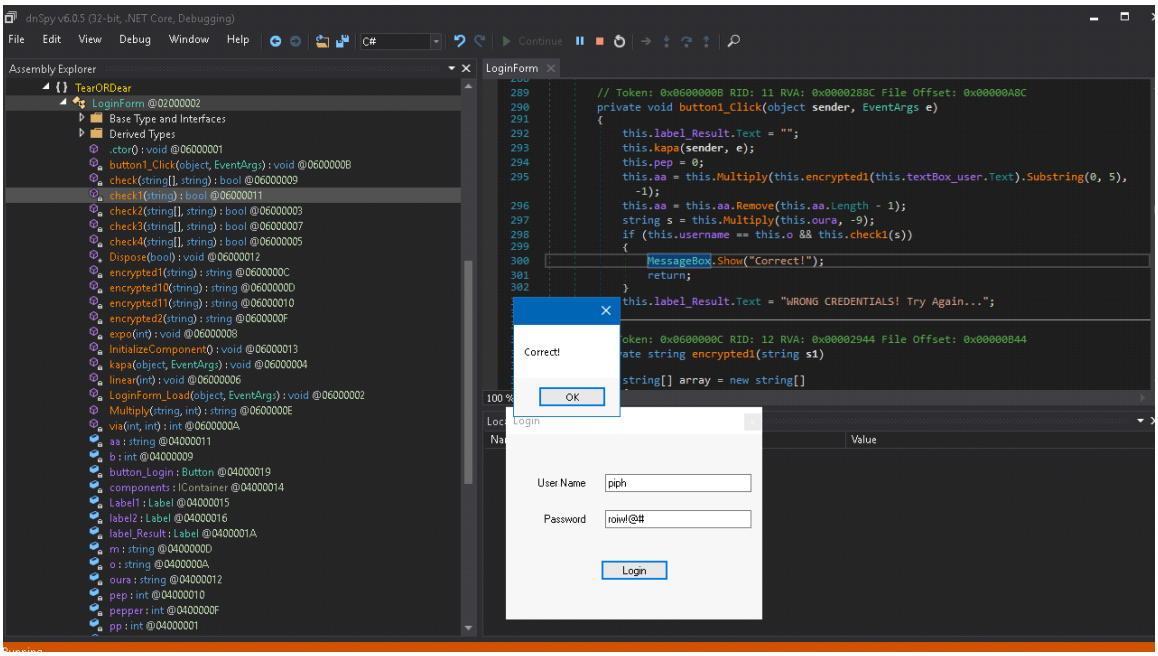
```
264                 110
265             };
266             int num = j + (int)(array[1] * array[5]) ^ 272;
267             if (type == 0)
268             {
269                 char.ToString((char)(j + 11));
270                 char c = (char)(array[2] - 101 + array[9] - 9);
271                 this.o += char.ToString(c);
272                 this.o += char.ToString(c);
273                 num += (Encoding.ASCII.GetBytes(this.o).Length + j + (int)(array[1] * array
                      [5]) ^ 272);
274             }
275             else if (type > 1)
276             {
277                 char.ToString((char)(j + 11));
278                 string str = char.ToString((char)(this.Src[2] + 12 + this.Src[10]));
279                 this.o += str;
280                 num += (Encoding.ASCII.GetBytes(this.o).Length + j + (int)(array[1] * array
                      [5]) ^ 272);
281             }
282             else
283             {
284                 num -= (Encoding.ASCII.GetBytes(this.o).Length + j + (int)(array[1] * array
                      [5]) ^ 272);
285             }
286             return num;
287         }
288
289         // Token: 0x0600000B RID: 11 RVA: 0x0000288C File Offset: 0x00000A8C
290         private void button1_Click(object sender, EventArgs e)
291         {
292             this.label_Result.Text = "";
293             this.kapa(sender, e);
```

Here we find variable o and now thru the code we hunt down what the value of this variable since by debugging we are able to basically see directly in the program. This is a good lesson as you can see in this program there are many rabbit holes without using this method you will

ultimately go down somewhere in the other functions that makes no sense so setting the breakpoint is critical. It is right at the comparison of the username string thats how you snuff out aa and find the first value.



Here you see 2 things and 1 to throw some possibly down a rabbit hole one is the password and the other is the encrypted value we see being compared in our username condition statement. To see if our theory and practice is on point we know try username = piph and there is now for o the password = roiw!@#



B I N G O !

So we now see we have our flag and now we can submit it and the program is reversed or cracked by extracting the values from the program while running. Important notes of working with breakpoints as by running the program and having source code insight it should be rather straight forward by working it variable by variable until you find the correct values.

C-3P0

RedWolfIntelligence