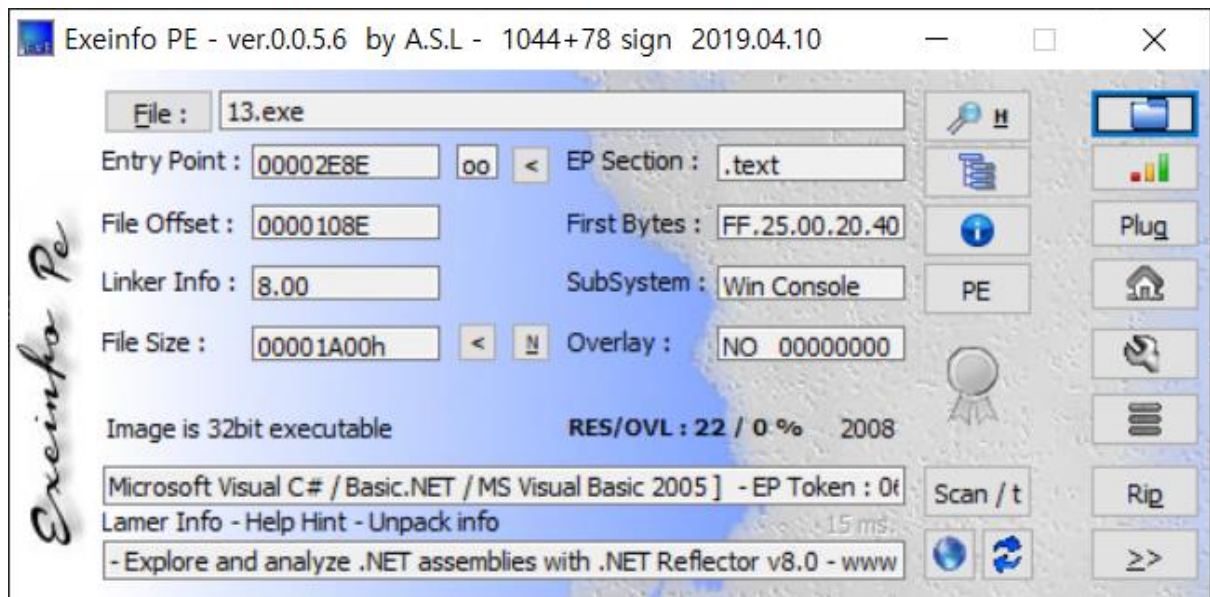


## 13.exe - 정답은 무엇인가

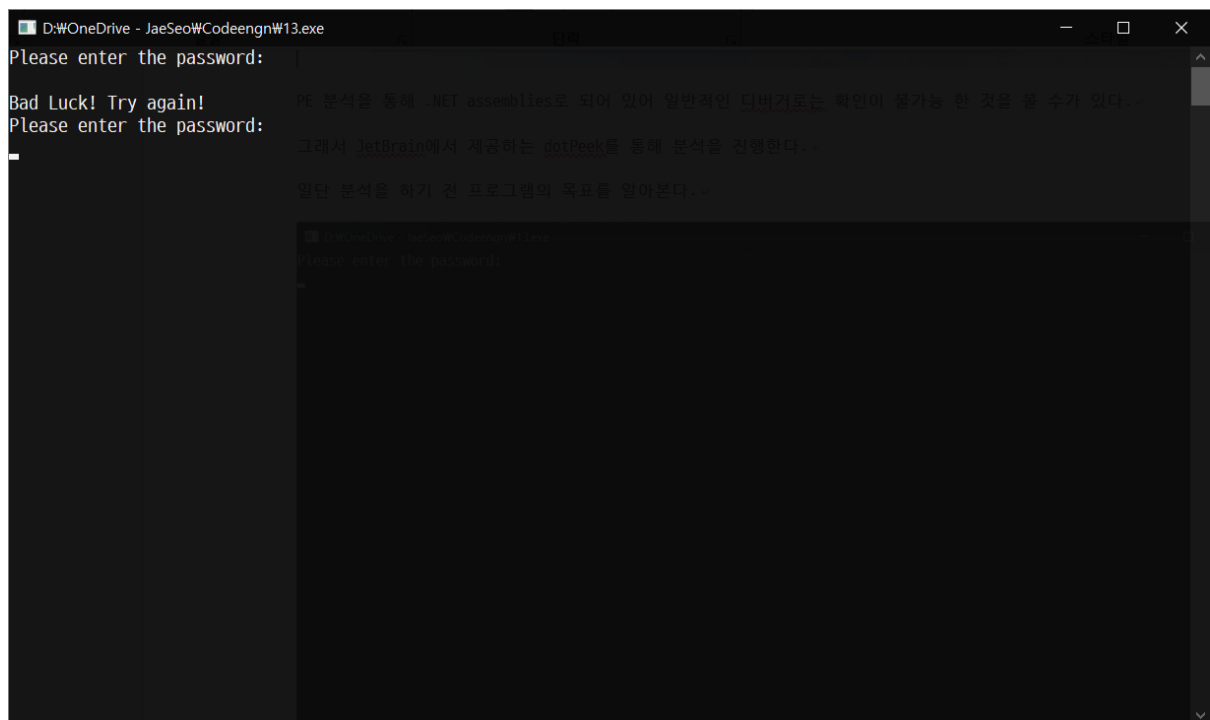
프로그램 PE 분석을 해본다.



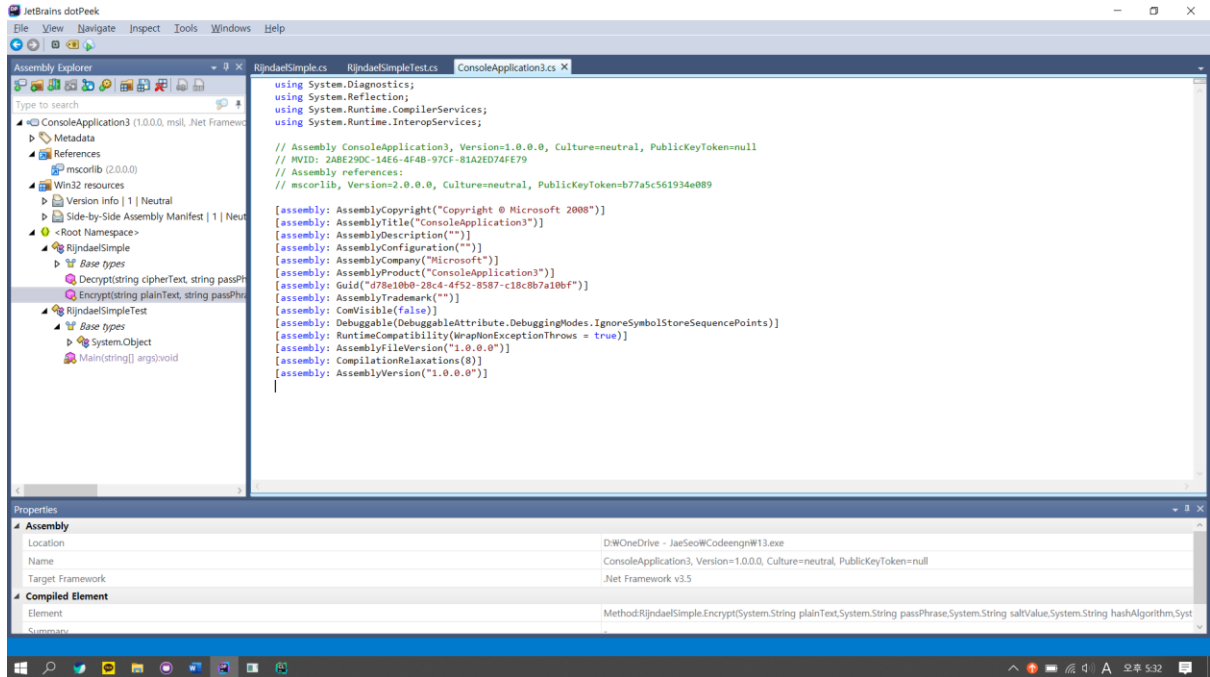
PE 분석을 통해 .NET assemblies로 되어 있어 일반적인 디버거로는 확인이 불가능 한 것을 볼 수가 있다.

그래서 JetBrains에서 제공하는 dotPeek를 통해 분석을 진행한다.

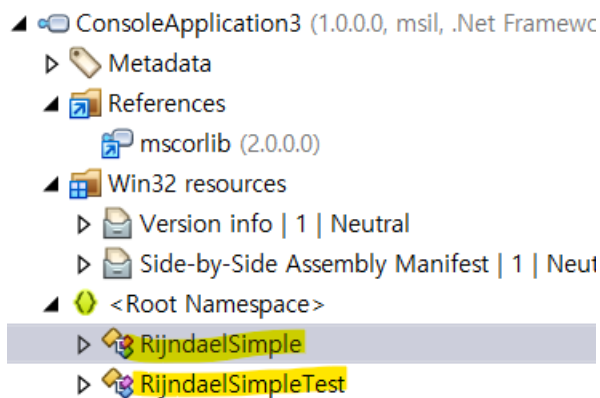
일단 분석을 하기 전 프로그램의 목표를 알아본다.



비번을 입력하고 입력 값이 올바른 지 체크하는 프로그램이다. 이제 dotPeek를 통해 열어서 확인한다.

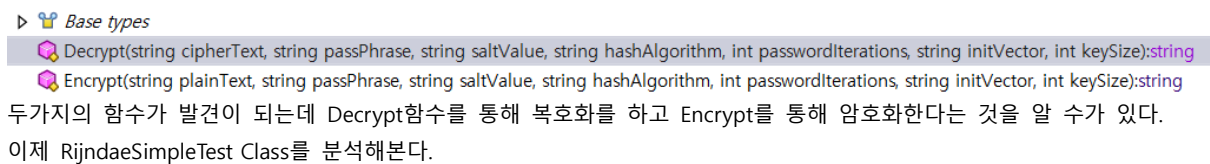


그 결과 위와 같이 분석이 되는 것을 확인할 수 있다.



이때 RijndaelSimple, RijndaelSimpleTest라는 두개의 Class를 발견하게 되는데 이 두개의 클래스를 들어가 분석을 해본다.

RijndaelSimple먼저 들어가 분석을 해본다.



이제 RijndaelSimpleTest Class를 분석해본다.

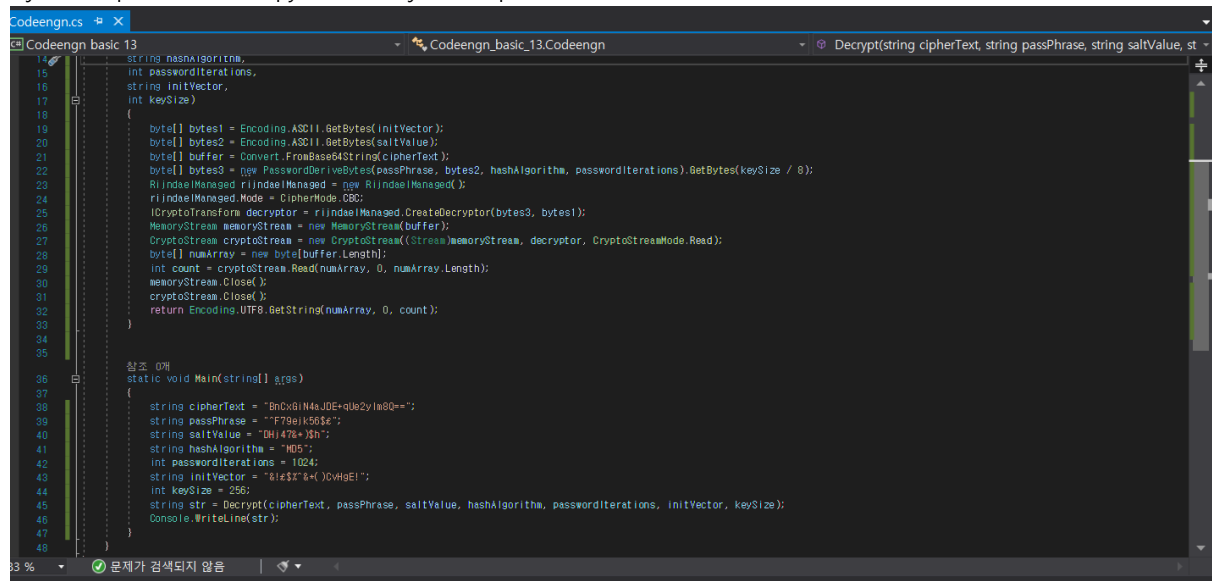
```

public class RijndaelSimpleTest
{
    [STAThread]
    private static void Main(string[] args)
    {
        string plainText = "";
        string cipherText = "BnCxCiN4aJDE+qUe2yIm8Q==";
        string passPhrase = "^F79ejk56$£";
        string saltValue = "DHj47&*)$h";
        string hashAlgorithm = "MD5";
        int passwordIterations = 1024;
        string initVector = "&!£$%^&*)CvHgE!";
        int keySize = 256;
        RijndaelSimple.Encrypt(plainText, passPhrase, saltValue, hashAlgorithm, passwordIterations, initVector, keySize);
        string str = RijndaelSimple.Decrypt(cipherText, passPhrase, saltValue, hashAlgorithm, passwordIterations, initVector, keySize);
        while (true)
        {
            Console.WriteLine("Please enter the password: ");
            if (!(Console.ReadLine() == str))
            {
                Console.WriteLine("Bad Luck! Try again!");
            }
            else
            {
                break;
            }
        }
        Console.WriteLine("Well Done! You cracked it!");
        Console.ReadLine();
    }
}

```

여기에 보면 Main이 있고 암호에 필요한 인자를 String에 저장하고 인자들을 가지고 복호화 하여 str에 저장하였다는 것을 볼 수가 있다. 또한 while문을 통해 입력한 값을 str과 비교하여 체크를 하는 모습을 볼 수 있다.

RijndaelSimple에 있는 Decrypt 함수와 RijndaelSimpleTest에 있는 인자 값을 가지고 C# 코딩을 하여 키값을 알아 낸다.

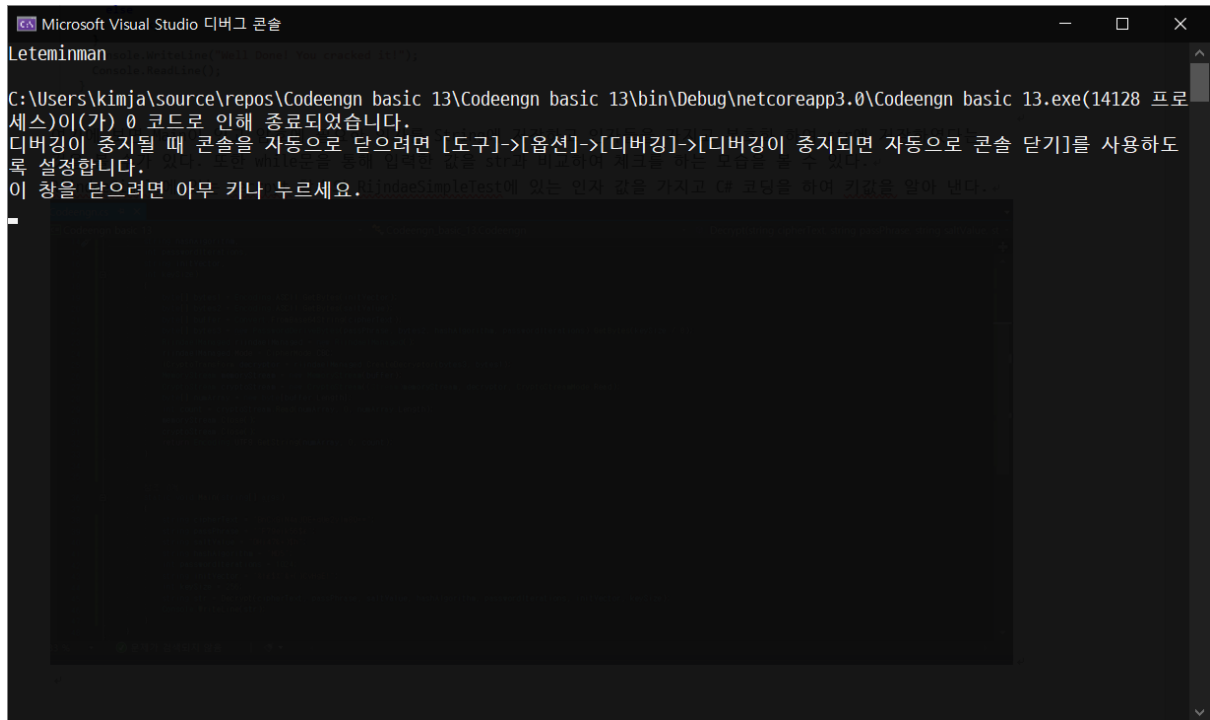


```

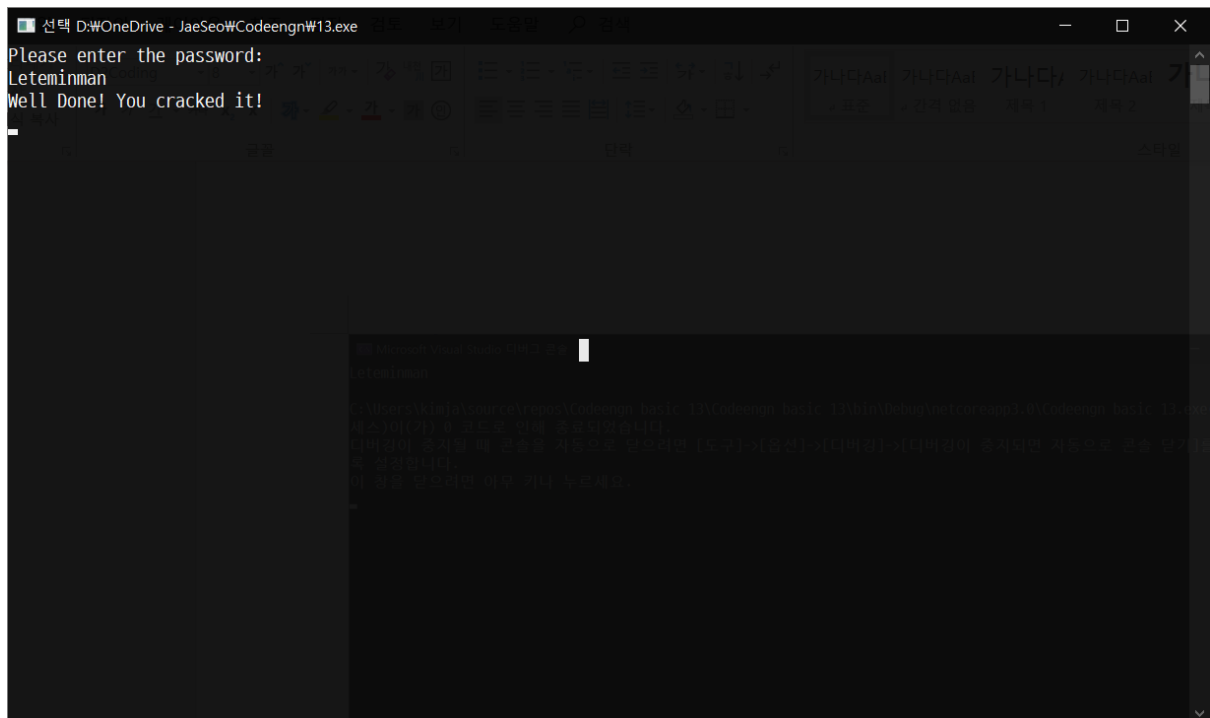
Codeengn.cs
Codeengn basic 13
Codeengn_basic_13.Codeengn
Decrypt(string cipherText, string passPhrase, string saltValue, st

14 string hashAlgorithm,
15 int passwordIterations,
16 string initVector,
17 int keySize)
18 {
19     byte[] bytes1 = Encoding.ASCII.GetBytes(initVector);
20     byte[] bytes2 = Encoding.ASCII.GetBytes(saltValue);
21     byte[] buffer = Convert.FromBase64String(cipherText);
22     byte[] bytes3 = new PasswordDeriveBytes(passPhrase, bytes2, hashAlgorithm, passwordIterations).GetBytes(keySize / 8);
23     RijndaelManaged rijndaelManaged = new RijndaelManaged();
24     rijndaelManaged.Mode = CipherMode.CBC;
25     ICryptoTransform decryptor = rijndaelManaged.CreateDecryptor(bytes3, bytes1);
26     MemoryStream memoryStream = new MemoryStream(buffer);
27     CryptoStream cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read);
28     byte[] numArray = new byte[buffer.Length];
29     int count = cryptoStream.Read(numArray, 0, numArray.Length);
30     memoryStream.Close();
31     cryptoStream.Close();
32     return Encoding.UTF8.GetString(numArray, 0, count);
33 }
34
35 참조 0개
36 static void Main(string[] args)
37 {
38     string cipherText = "BnCxCiN4aJDE+qUe2yIm8Q==";
39     string passPhrase = "^F79ejk56$£";
40     string saltValue = "DHj47&*)$h";
41     string hashAlgorithm = "MD5";
42     int passwordIterations = 1024;
43     string initVector = "&!£$%^&*)CvHgE!";
44     int keySize = 256;
45     string str = Decrypt(cipherText, passPhrase, saltValue, hashAlgorithm, passwordIterations, initVector, keySize);
46     Console.WriteLine(str);
47 }
48
83 %
문제가 검색되지 않음

```



그결과 Leteminman 라는 평문이 나오는데 이것을 프로그램에 넣어 직접 테스트를 해본다.



정답: Leteminman