

Ochrona Danych Osobowych i Technologie
Wzmacniające Prywatność
Laboratorium nr 6

Julia Sadecka, Jakub Augustyn, Benjamin Jankowski

Kompilacja i dekompilacja - plik .exe, .dll, .pdb

Wybraliśmy kod gry „zgadnij liczbę”, dostępny poniżej:

```
program.cs
1  using System;
2  namespace NumberGuesser
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              GetAppInfo(); // Run GetAppInfo function to get info
9              GreetUser(); // Ask for users name and greet
10             while (true)
11             {
12                 // Create a new Random object
13                 Random random = new Random();
14                 // Init correct number
15                 int correctNumber = random.Next(1, 10);
16
17                 int guess = 0;
18                 Console.WriteLine("Guess a number between 1 and 10");
19                 while (guess != correctNumber)
20                 {
21                     string input = Console.ReadLine();
22
23                     // Make sure its a number
24                     if (!int.TryParse(input, out guess))
25                     {
26                         // Print error message
27                         PrintColorMessage(ConsoleColor.Red, "Please use an actual number");
28
29                         continue;
30                     }
31
32                     guess = Int32.Parse(input);
33
34                     // Match guess to correct number
35                     if (guess != correctNumber)
36                     {
37                         PrintColorMessage(ConsoleColor.Red, "Wrong number, please try again");
38                     }
39                 }
40
41                 // Print success message
42                 PrintColorMessage(ConsoleColor.Yellow, "CORRECT!! You guessed it!");
43
44                 Console.WriteLine("Play Again? [Y or N]");
45
46                 // Get answer
47                 string answer = Console.ReadLine().ToUpper();
48
49                 if (answer == "Y") {
50                     continue;
51                 }
52                 else if (answer == "N") {
53                     return;
54                 }
55                 else
56                 {
57                     return;
58                 }
59             }
60         }
61     }
62 }
```

Aby uruchomić, a później również skompilować kod należało:

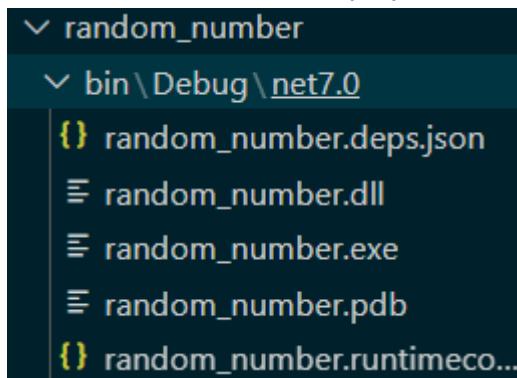
1. Pobrać .NET ze strony microsoftu (używana wersja to:)

```
PS C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO> dotnet --version
7.0.203
```

2. Utworzyć nowy projekt komendą `dotnet new console -o random_number` - gdzie flaga `-o` tworzy nam docelowy folder projektu
3. Następnie w pliku `Program.cs` możemy podmienić kod na kod naszej gry
4. Aby odpalić plik wystarczy wpisać `dotnet run` (gdy jesteśmy już w folderze docelowym projektu)

```
PS C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number> dotnet run
C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\Program.cs(21,36): warning CS8600: Converting null literal or possible null value to no
n-nullable type. [C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\random_number.csproj]
C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\Program.cs(47,33): warning CS8602: Dereference of a possibly null reference. [C:\Users\
Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\random_number.csproj]
C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\Program.cs(86,32): warning CS8600: Converting null literal or possible null value to no
n-nullable type. [C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\random_number.csproj]
Number Guesser: Version 1.0.0 by Brad Traversy
What is your name?
```

5. Skompilowane pliki pokażą się w folderze `/bin/Debug/net7.0`:



Zamiast odpalić grę możemy również skorzystać z komendy: `dotnet build`

```
PS C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number> dotnet build
MSBuild version 17.5.1+f6fdcf537 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
random_number -> C:\Users\Benek\OneDrive\Pulpit\Studia_AGH\Sem_IV\ODO\random_number\bin\Debug\net7.0\random_number.dll

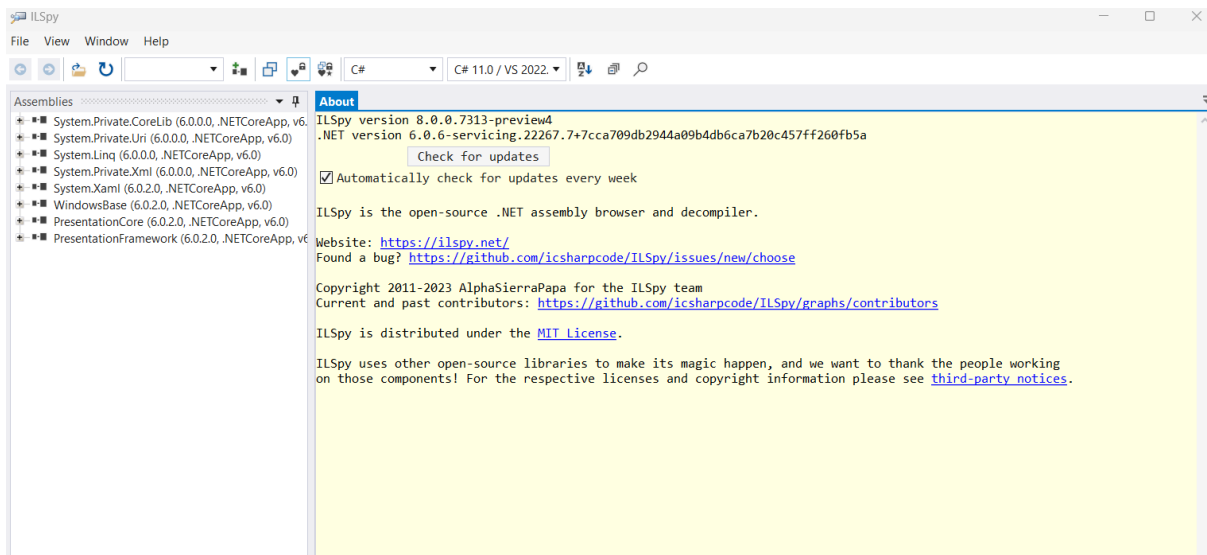
Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.99
```

2. Proces dekompilacji plików

- A) Pierwszy sposób - można zainstalować osobną aplikację ILSpy pobraną z githuba (<https://github.com/icsharpcode/ILSpy/releases>)

Tak wygląda odpalony program (plik ILSpy.exe):



Poniżej został wklejony zdekompilowany kod:

```
// random_number, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null
// NumberGuesser.Program
using System;

internal class Program
{
    private static void Main(string[] args)
    {
        GetAppInfo();
        GreetUser();
        string answer;
        do
        {
            Random random = new Random();
            int correctNumber = random.Next(1, 10);
            int guess = 0;
            Console.WriteLine("Guess a number between 1
and 10");
            while (guess != correctNumber)
            {
                string input = Console.ReadLine();
```

```

        if (!int.TryParse(input, out guess))
        {
            PrintColorMessage(ConsoleColor.Red,
"Please use an actual number");
            continue;
        }
        guess = int.Parse(input);
        if (guess != correctNumber)
        {
            PrintColorMessage(ConsoleColor.Red,
"Wrong number, please try again");
        }
    }
    PrintColorMessage(ConsoleColor.Yellow,
"CORRECT!! You guessed it!");
    Console.WriteLine("Play Again? [Y or N]");
    answer = Console.ReadLine().ToUpper();
}

while (answer == "Y");
if (!(answer == "N"))
{
}

}

private static void GetAppInfo()
{
    string appName = "Number Guesser";
    string appVersion = "1.0.0";
    string appAuthor = "Brad Traversy";
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("{0}: Version {1} by {2}",
appName, appVersion, appAuthor);
    Console.ResetColor();
}

```

```

private static void GreetUser()
{
    Console.WriteLine("What is your name?");
    string inputName = Console.ReadLine();
    Console.WriteLine("Hello {0}, let's play a
game...", inputName);
}

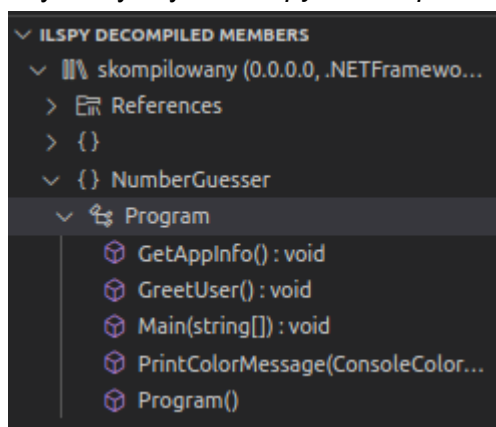
private static void PrintColorMessage(ConsoleColor
color, string message)
{
    Console.ForegroundColor = color;
    Console.WriteLine(message);
    Console.ResetColor();
}
}

```

Jako można zauważyć sam kod aplikacji niewiele różni się od oryginalnego kodu - sposób działania jest taki sam, występują tylko nieco inne rozwiązania np. zamiast pętli *while(true)* -> *do {...} itd.*

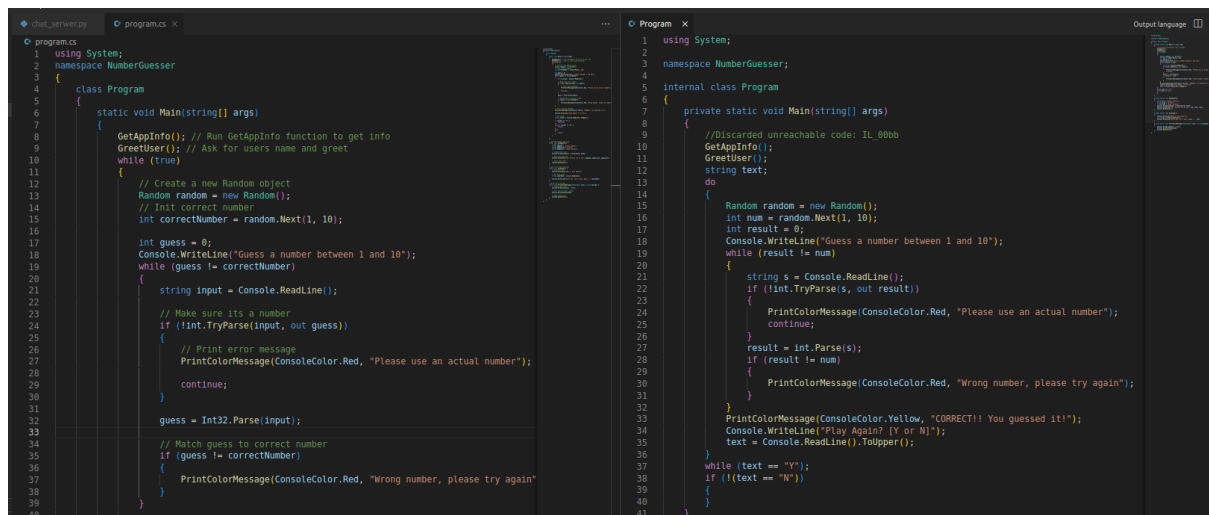
B) Wtyczka bezpośrednio zainstalowana w Visual Studio Code

Używamy wtyczki *ILSpy Decompiled Members* do oprogramowania Visual Studio Code:



Jak widać, kroki przebiegły pomyślnie bez błędów po drodze.

Zestawienie programu początkowego i zdekompilowanego widoczne jest poniżej:



Wnioski Kompilacja:

Jak można zauważyć, różnice w kodzie występują, lecz nie mają one wpływu na działanie - pełna mechanika programu została zachowana bez uszczerbku.

Zaletą jest szczególnie brak widocznych wcześniej komentarzy. Gwarantuje to, że myśli pozostawione przez autora w kodzie nie wyjdą na światło dzienne.

Obfuskacja kodu

Dokonuję zaciemnienia kodu aplikacji za pomocą programu *dotNET Reactor*. Operacja przebiega pomyślnie:

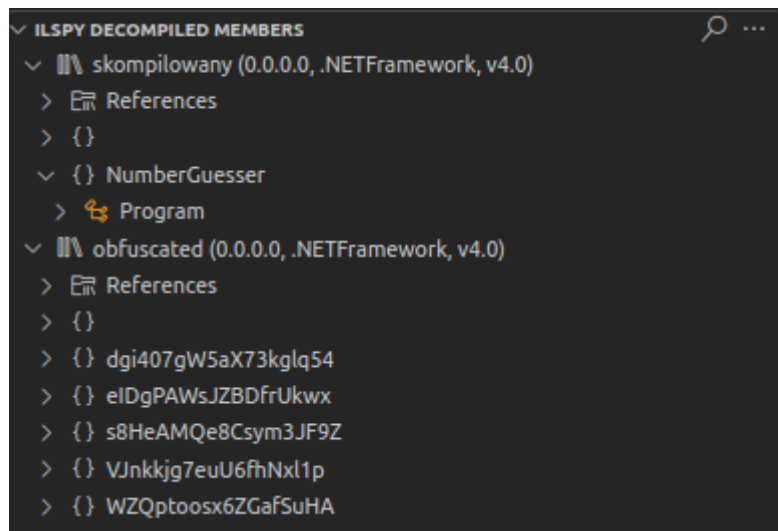
```
kuba@lenovo:~/Downloads$ ./dotNET_Reactor -obfuscation 1 -file ~/sem_4/odo/skompilowany.exe

.NET Reactor - Analyzing Assemblies ...

.NET Reactor - Protecting skompilowany ...
---- skompilowany ----
Examining Code...
Done
Suppress Decompilation - Step 1...
Done
Obfuscation (Naming Convention -> Standard)...
Done
String Encryption...
Done
Suppress Decompilation - Step 2...
Done
Final Steps...
Done

skompilowany - Successfully Protected!
```

Po dekompilacji widać pierwsze różnice: wcześniejsze nazwy zostały zastąpione losowym ciągiem znaków:



Ponadto aplikacja zaciemniajaca dodała „coś od siebie”:

```
1 using System;
2
3 internal static void AKVLp0naej()
4 {
5     //Discarded unreachable code: IL_0002
6     if (!CQ3g80nkuJ)
7     {
8         CQ3g80nkuJ = true;
9         if (Math.Abs((DateTime.Now - new DateTime(2023, 4, 15)).Days) >= 14)
10        {
11            throw new Exception("This assembly is protected by an unregistered version of Eziriz's \".NET Reactor\"! This assembly won't further work.");
12        }
13    }
14 }
15
```

Porównanie aplikacji - kod pierwotny (po prawej) oraz poddany obfuskacji i dekompilacji:

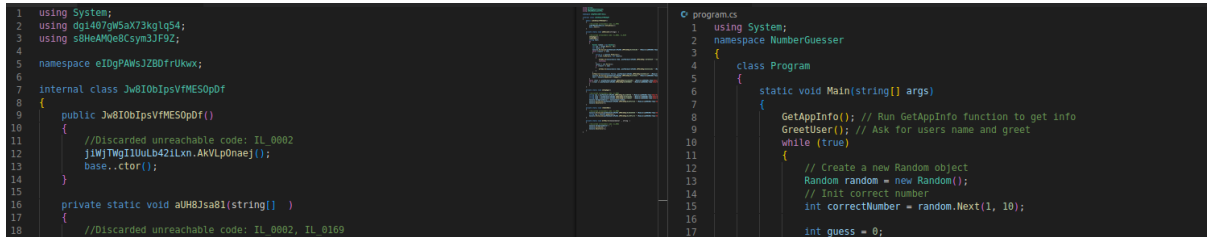
The image shows a side-by-side comparison of two code snippets. On the left is the original code, which is heavily obfuscated with names like 's8HeAMQe8Csym3JF9Z', 'aUH83a81(string[])', and 'Random random = new Random();'. On the right is the decompiled code, which is much more readable and shows the logic of a number guessing game. The decompiled code includes a 'Main' method that initializes a 'Random' object, generates a 'correctNumber', and enters a loop where it prompts the user to guess a number between 1 and 10. It provides feedback on whether the guess is correct or incorrect.

Można dostrzec w zaciemnionym kodzie odpowiadające mu fragmenty pierwotnie, jednak jest on bardzo nieczytelny dla ludzkiego oka - używane nazwy zmiennych są bardzo dezorientujące.

Mylące może być też dodanie biblioteki o nazwie która nic nam nie mówi.

Jedynym niezmiennym elementem są funkcje jak np. *Random*, *Console.WriteLine*, ponieważ są to funkcje typowo używane w danym języku programowania. Jednak analiza statyczna powyższego kodu tak naprawdę nie nasuwa żadnych wniosków.

Ponadto zastanawiające są zakomentowane fragmenty *Discarded Unreachable code*:



```
1 using System;
2 using d6i407dW5aX73kg1q54;
3 using s8HeAMQe8Csym3jF9Z;
4
5 namespace eIDgPAWsJZB0frUkwx;
6
7 internal class Jw8IObIpsVFME5OpDf
8 {
9     public Jw8IObIpsVFME5OpDf()
10     {
11         //Discarded unreachable code: IL 0002
12         jIWjTWgIIUhLb42iLxn.AkVLP0naej();
13         base..ctor();
14     }
15
16     private static void aUHB3sa8l(string[] )
17     {
18         //Discarded unreachable code: IL 0002, IL 0169
19     }
20 }
```

```
1 using System;
2 namespace NumberGuesser
3 {
4     class Program
5     {
6         static void Main(string[] args)
7         {
8             GetAppInfo(); // Run GetAppInfo function to get info
9             GreetUser(); // Ask for users name and greet
10             while (true)
11             {
12                 // Create a new Random object
13                 Random random = new Random();
14                 // Init correct number
15                 int correctNumber = random.Next(1, 10);
16
17                 int guess = 0;
```

Mogą być one pozostałościami po komentarzach, lecz występują także w miejscach, gdzie komentarzy nie było.

Jest to cecha obfuskacji, gdyż komentarze pozostawione przez autora w niepowołanych rękach mogłyby zdradzać strukturę programu lub pliku z nim współpracującego

Wnioski:

Kod niepoddany obfuskacji po procesie dekompilacji jest tak samo czytelny jak pierwotny. Brak w nim komentarzy, jednak jak powszechnie wiadomo - mimo że pomagają, nie one są najważniejsze w kodzie.

Obfuskacja natomiast praktycznie odbiera możliwość odtworzenia oryginalnego kodu.

Otrzymane pliki wyglądają chaotycznie, nazwy zmiennych, funkcji a nawet wypisywanych *stringów* są praktycznie nieczytelne dla ludzkiego oka. Ponadto zmienia się struktura kodu, co także utrudnia potencjalnemu atakującemu odtworzenie pierwotnego kodu twórcy.

Minusem obfuskacji - ze względu np. na długość nazw - może być spowolnienie wykonywanego programu, zwiększenie potrzebnego miejsca na dysku, czy też w końcu może to doprowadzić do błędów w działaniu.