.NET executables - internals & analyzing

Ing. Martin Jirkal

Outline

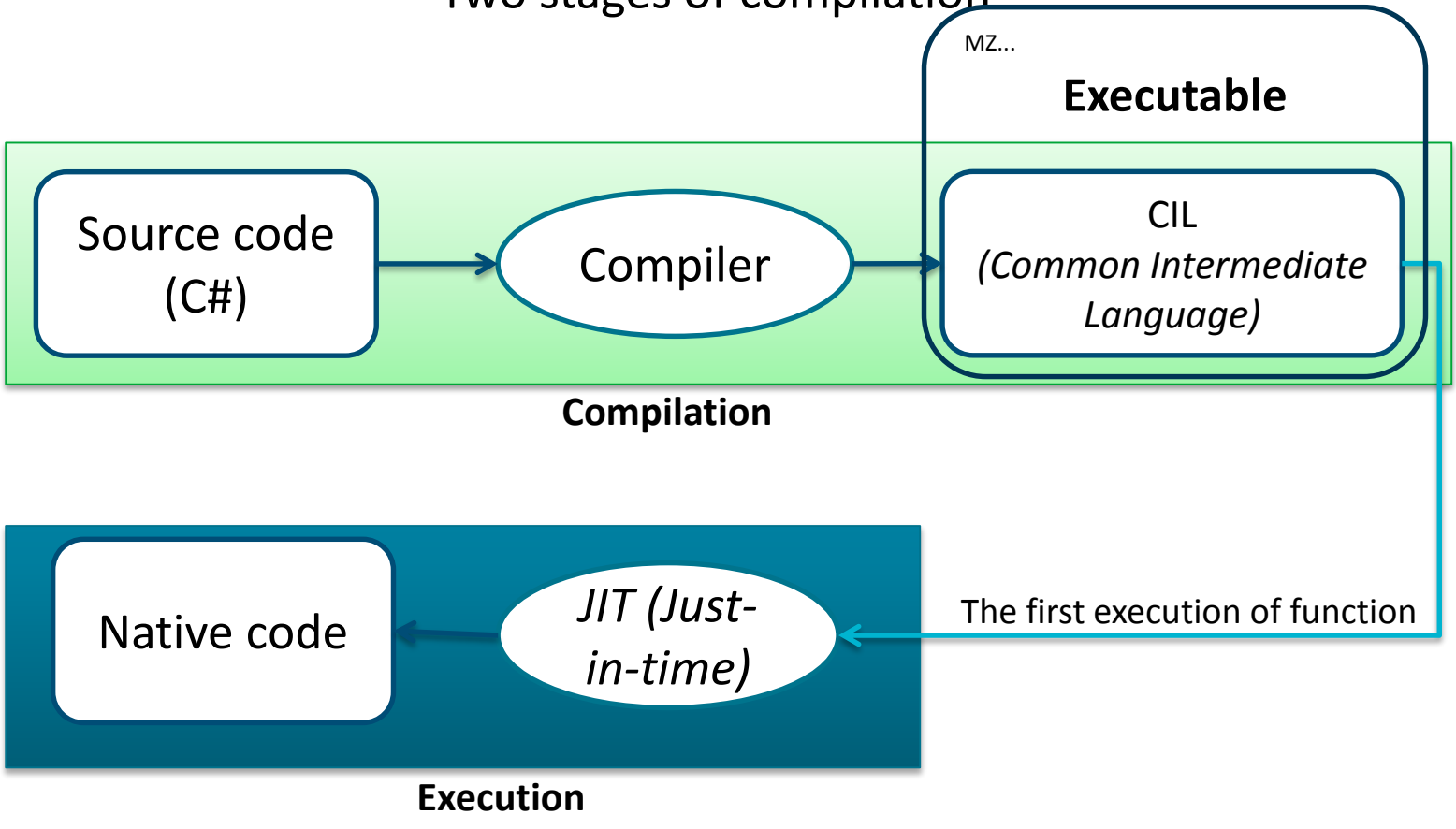
- .NET executable format
- Analyzing
- MSIL protectors

.NET executable format

.NET platform, MSIL, CIL, CLR, Managed (by runtime), …

- Two stage of compilation
- CIL
- Extension of PE format
- Object-oriented
- User symbols in executable

Two stages of compilation



CIL

- formerly called MSIL
  - 200+ opcodes
  - Stack machine
- “High-level assembler”
  - Works on object

```
// bigTestSample.utils
public static int add(int a, int b)
{
    return a + b;
}
```

C#

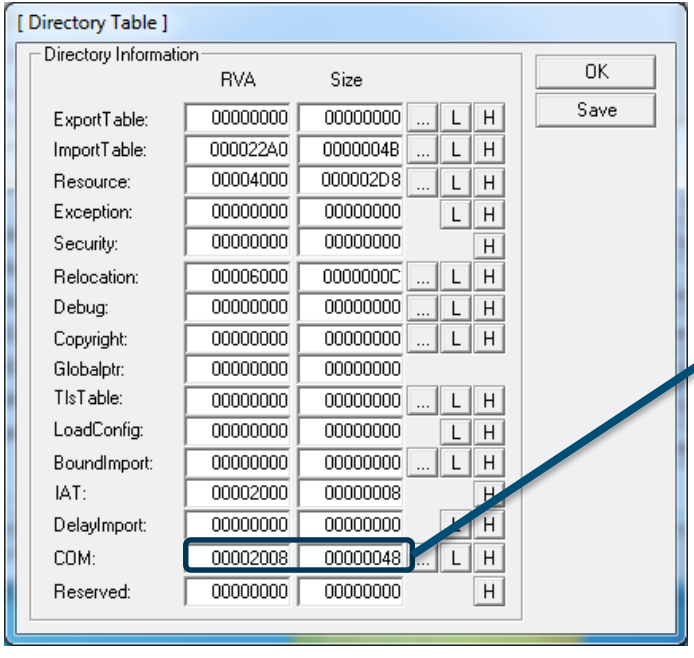
```
.method public hidebysig static
    int32 'add' (
        int32 a,
        int32 b
    ) cil managed
{
    // Method begins at RVA 0x23c8
    // Code size 9 (0x9)
    .maxstack 2
    .locals init (
        [0] int32
    )

    IL_0000: nop
    IL_0001: ldarg.0
    IL_0002: ldarg.1
    IL_0003: add
    IL_0004: stloc.0
    IL_0005: br.s IL_0007

    IL_0007: ldloc.0
    IL_0008: ret
} // end of method utils::'add'
```

CIL

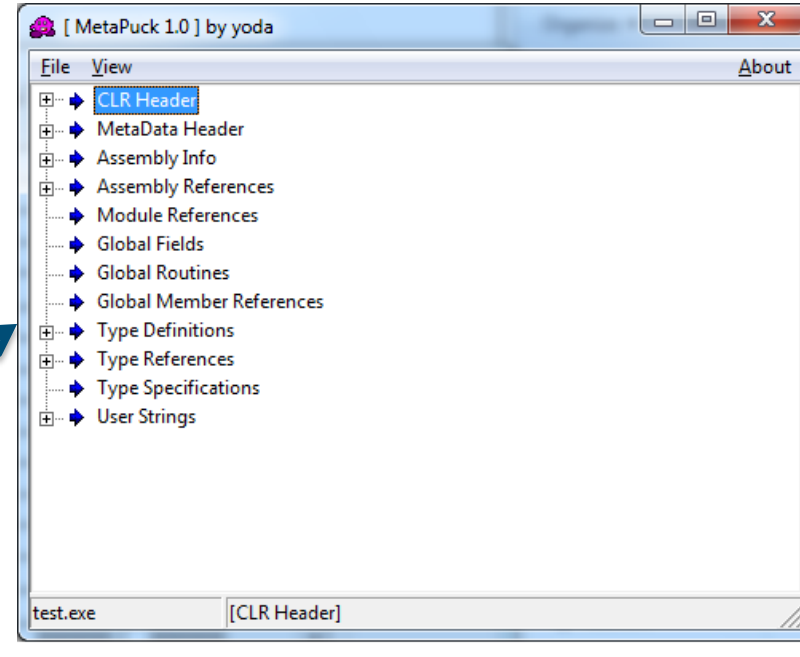
Tool: ILSpy

Extension of PE format

PE Optional Header



CLR (.NET) Header

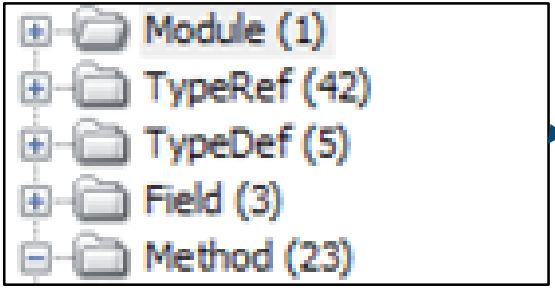


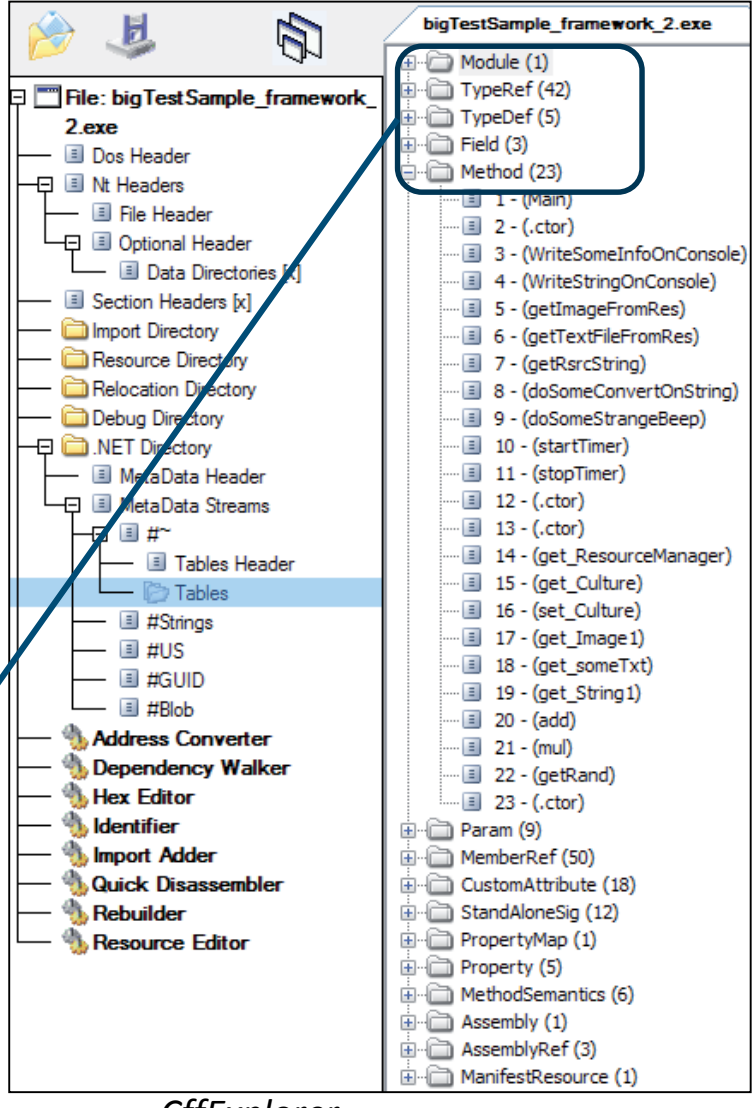
LordPE

Object-oriented

Program structure is kept in executable:

- Object (classes, methods, fields) in Metadata **tables**
- Referenced with tokens in CIL



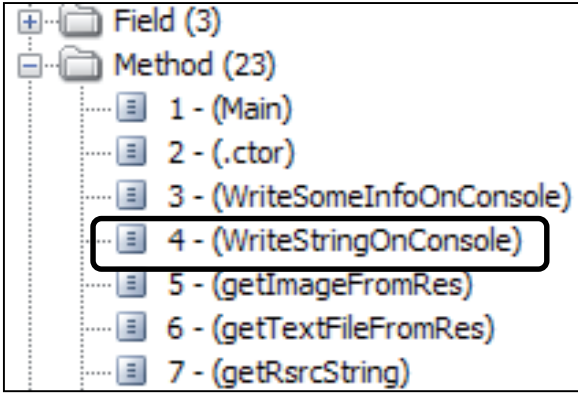


CffExplorer

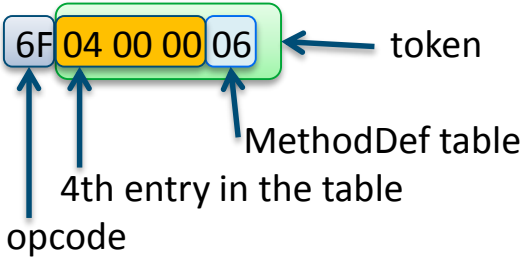
Object-oriented

Program structure is kept in executable:

- Object (classes, methods, fields)
  in Metadata tables
- Referenced with **tokens** in CIL




```
IL_0026: callvirt instance void bigTestSample.TestCases::WriteStringOnConsole(string)
L_00000026:          6F 04 00 00 06          callvirt 0x06000004
```



User symbols in executable

In .NET names of classes, methods and fields are kept in executable.

It is like analyzing native x86 file with .pdb

(.pdb - Program Database file – store debug informations)

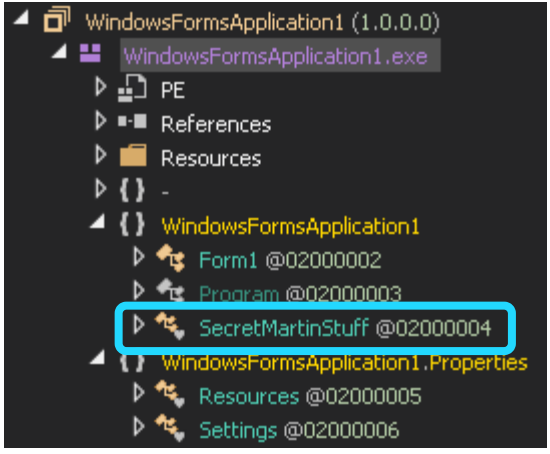
(but without .pdb !!!)

User symbols in executable

Example:

- CV 11

- Contains executable in resources
  - Looks like Crackme



dnSpy

User symbols in executable

Other examples of method names from real-world samples :

- *ShowTrialModeMessageBox()*
- *CheckLicence()*
- *Keylogger()*

As C# developer:

- Be aware of keeping your symbols in executable
- Or use obfuscator *

.NET analysis

- Static analysis
- Dynamic analysis

Static analysis

- CFF Explorer (CLR Metadata tables)
- DnSpy, ILSpy (format, decompiler)

```
.method public hidebysig static
    int32 'mul' (
        int32 a,
        int32 b
    ) cil managed
{
    // Method begins at RVA 0x23e0
    // Code size 9 (0x9)
    .maxstack 2
    .locals init (
        [0] int32
    )

    IL_0000: nop
    IL_0001: ldarg.0
    IL_0002: ldarg.1
    IL_0003: mul
    IL_0004: stloc.0
    IL_0005: br.s IL_0007

    IL_0007: ldloc.0
    IL_0008: ret
} // end of method utils::'mul'
```

Decompilation

```
// bigTestSample.utils
public static int mul(int a, int b)
{
    return a * b;
}
```

ILSpy

Dynamic analysis - Debugging

Sometimes we must go deeper...

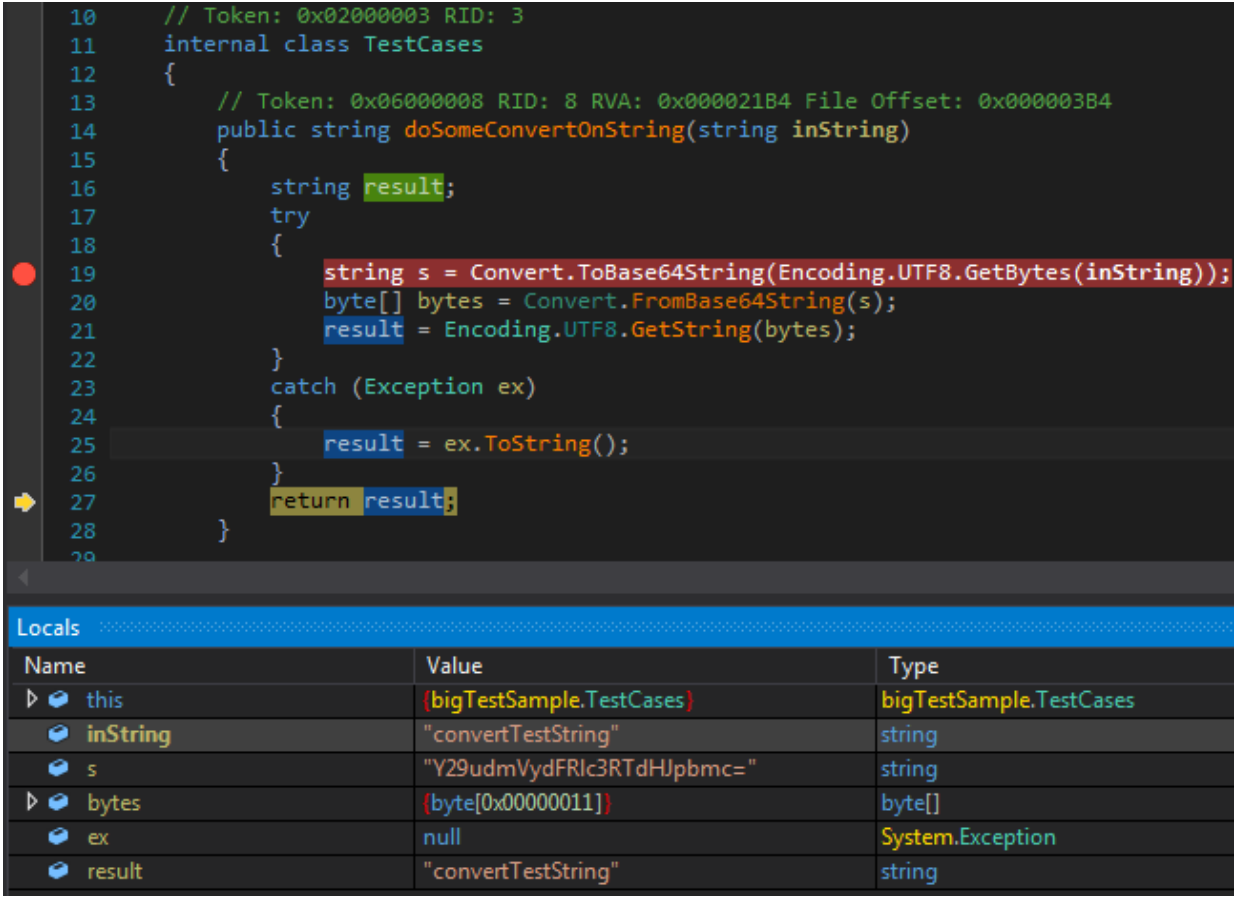
... but with MSILs it is not so easy...

… because there are two stages of compilation :/

dnSpy

- Breakpoints
- Trace in CIL or C#
- moveOrigin
- localVars

- openSource



WinDbg

.NET plugins:

- SOS.dll
- SOSEX.dll

„*Unpack your troubles*: .NET packer tricks and countermeasures*"

Hartung, VB2015, Prague

Other debuggers

- ILSpy debugger (plugin)
- PeBrowseDbg
- ...

Work only sometimes, limited functionality.

.NET protectors & obfuscators

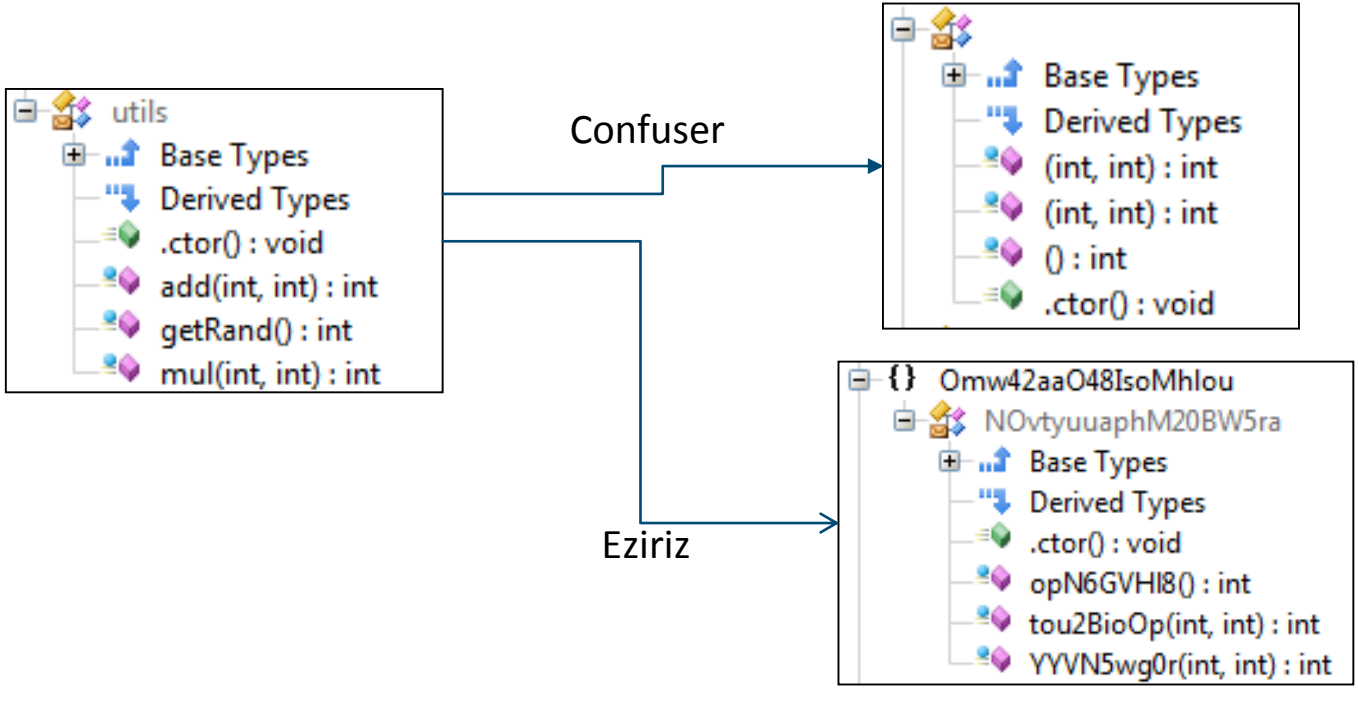
Make RE more challenging ☺

- Strip debug symbols
- UserString's obfuscation
- Layers hiding (Assembly.Load())
- Hide CIL code
- Program flow obfuscation
- Hinder or block decompilation
- ...

Strip debug symbols

„Clean” file

„Obfuscated” file



UserString's obfucation

```
ldstr ".net testSample - console version - v1.0\n\n"
```

What packers do with UserStrings?

```
ldc.i4 24
call string Q6vMeDahMxaSGoE1So.RRPguRkAG3Y43m0LA2::Njg7MFDG6$PST06000017(int32)
```

```
ldc.i4 -657940626
call !!0 '<Module>'::''<string>(uint32)
```

```
ldstr "▯▯▯▯▯▯▯▯▯▯"
ldc.i4 62823
call string '▯'::'▯'(string, int32)
```

Every *ldstr* opcode is changed into call to decrypt method.

Packer – next layer loading

- Like old-fashioned native packer - decrypt & execute next layer
  - .NET has special API – **Assembly.Load()**

```
byte[] rawAssembly;
using (MemoryStream memoryStream = new MemoryStream())
{
    manifestResourceStream.CopyTo(memoryStream);
    rawAssembly = memoryStream.ToArray();
}
Assembly assembly = Assembly.Load(rawAssembly);
Console.WriteLine("Assembly loaded\n");
assembly.EntryPoint.Invoke(null, new object[]
{
    args
});
Console.WriteLine("Assembly invoked\n");
```

(some packers, bladabindi malware family, ...)

wikipedia.org

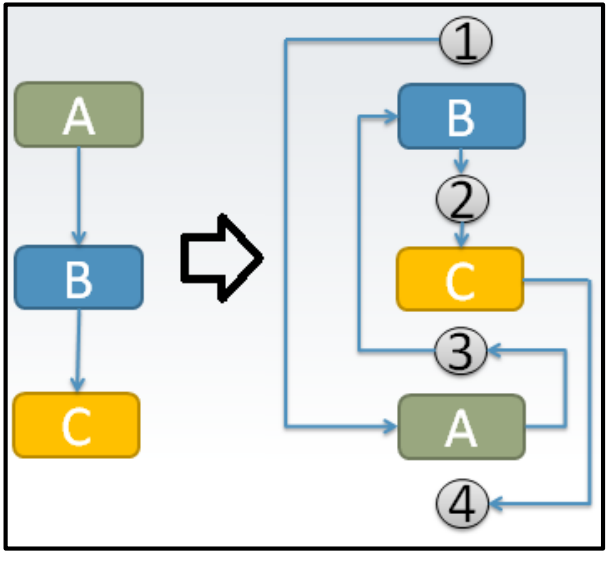
Hide CIL code

Call to decrypt user code method from <Module>::.cctor()

```
.class private auto ansi '<Module>'
{
    // Methods
    .method private hidebysig specialname rtspecialname static
        void .cctor () cil managed
    {
        // Method begins at RVA 0x2050
        // Code size 6 (0x6)
        .maxstack 8

        IL_0000: call void eDyvLNuENDt5Xi1U2C::ewwmj81c4$P25()
        IL_0005: ret
    } // end of method '<Module>'::.cctor

} // end of class <Module>
```

Program flow obfuscation

Make new basic-blocks and mix it:



Decompiled code looks worse