UNPACK YOUR TROUBLES: .NET PACKER TRICKS AND COUNTERMEASURES

Marcin Hartung

ESET, Poland

UNPACK YOUR TROUBLES: .NET PACKER TRICKS AND COUNTERMEASURES

Marcin Hartung

hartung@eset.pl

Eset Poland

At Eset: programmer in the Software Protectors Analysis & Unpacking Team

Other: motorbiker, mountain hiker, climber

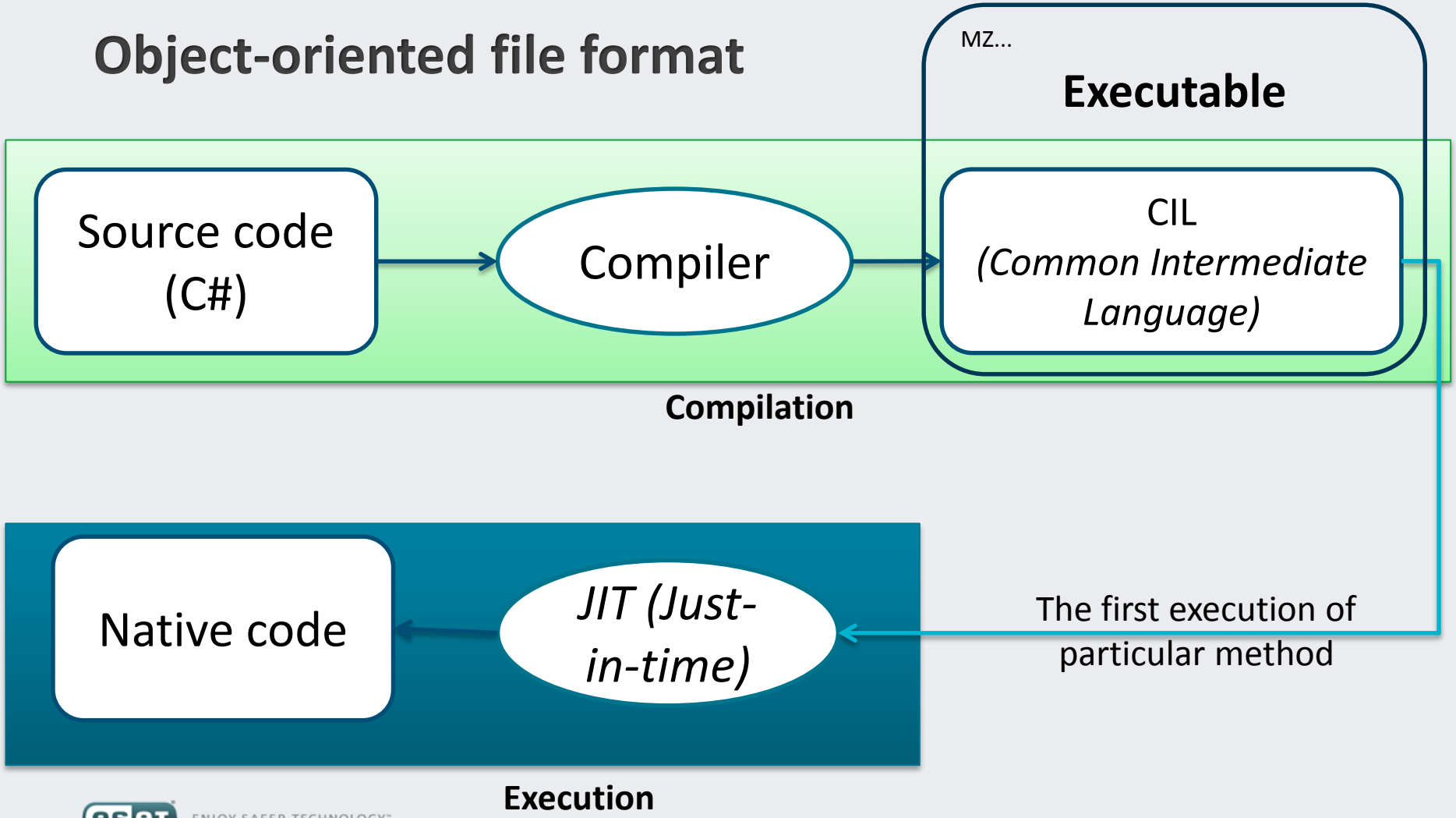
Outline

- Object-oriented file format

- .NET analysing

- LoadAssembly
- UserString
- API

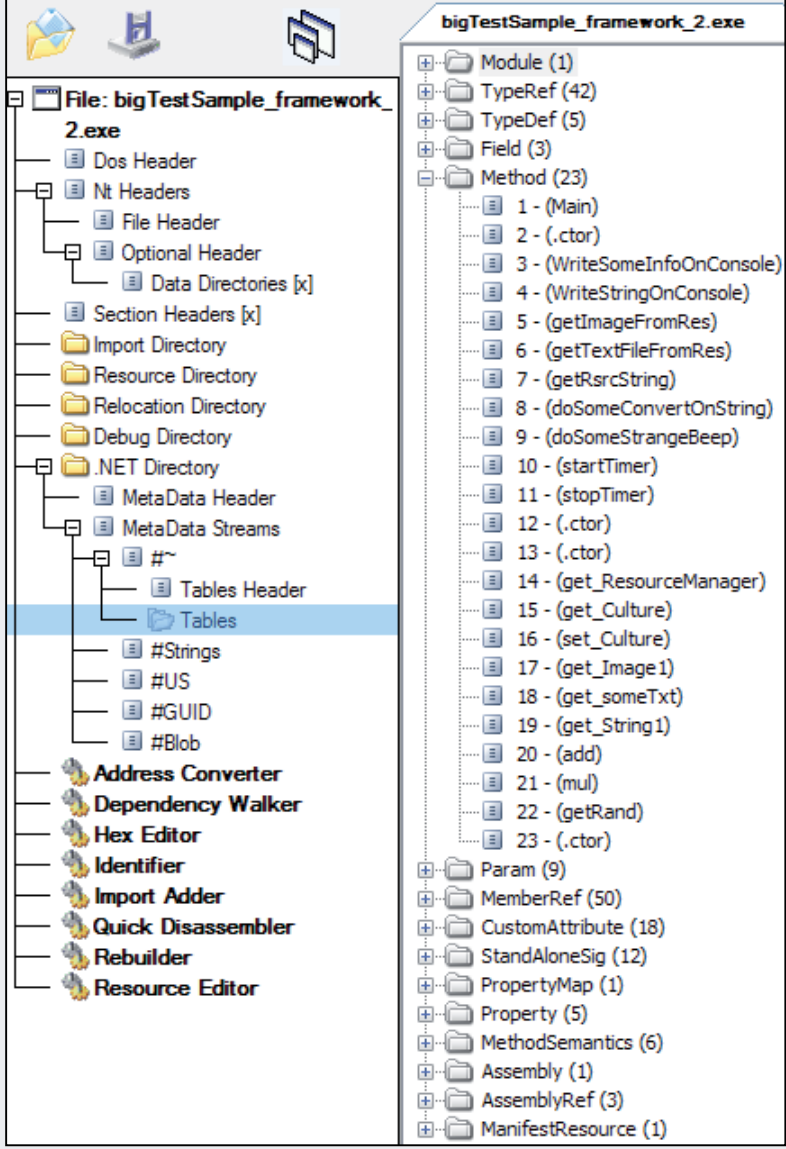
Object-oriented file format



Object-oriented file format

Program structure is kept in executable:

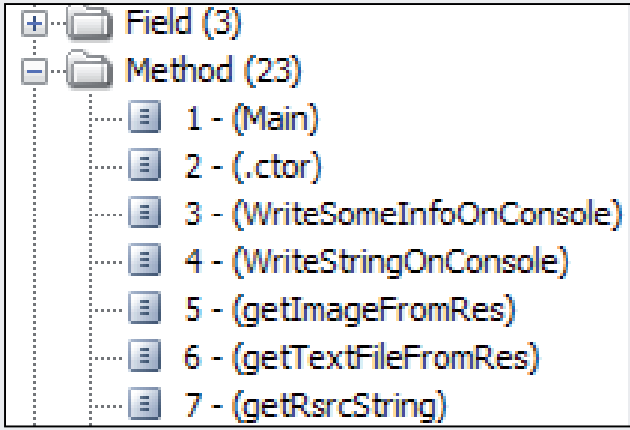
- Object (classes, methods, fields) in tables
- Referenced with tokens in CIL
- User method names



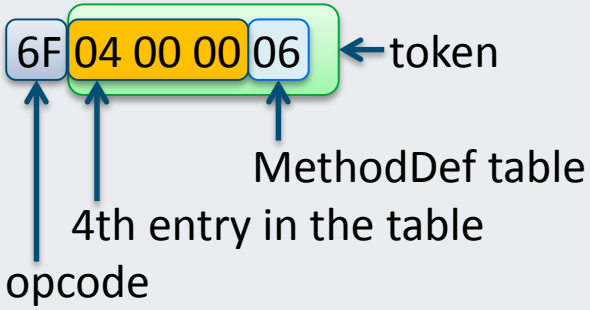
Object-oriented file format

Program structure is kept in executable:

- Object (classes, methods, fields) in tables
- Referenced with tokens in CIL
- User method names




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



Analysing .NET samples

- Static analysis (decompilers)
  - good for non-protected samples,
  - bad (or imposible) for obfuscation

- Deobfuscators
  - support for every new version
  - problem with patched or custom packers

Analysing .NET samples

- Debugging

  symbols, runtime sources

  winDbg + sos.dll plugin (*!bpmd, !dumpmd*)

- Other

  emulating, profiling, ...

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

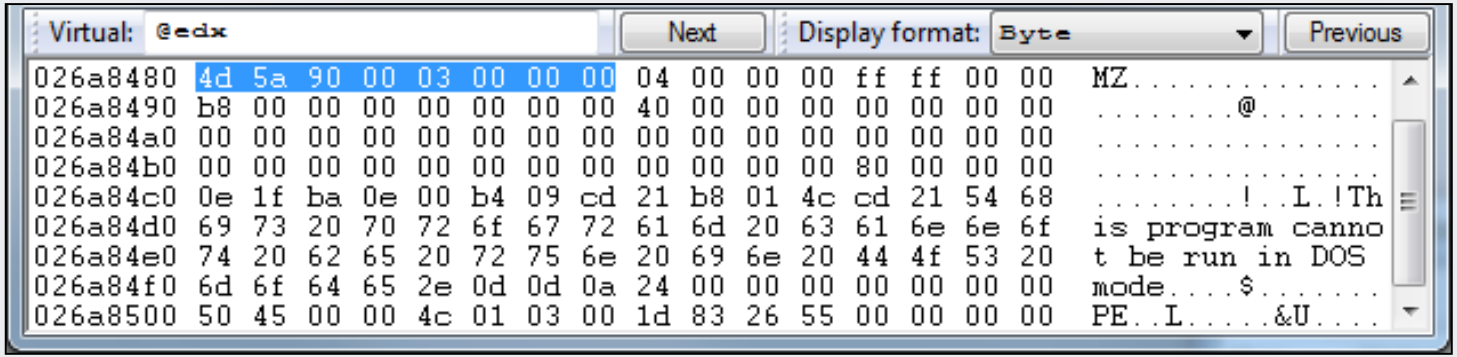
Packer – next layer loading

Assembly.Load() – solution – catching next layer MZ during loading

```
bp mscorwks!CLRMapViewOfFileEx + 0x26 "da eax" /
bp clr!AssemblyNative::LoadFromBuffer "dd (edx - 4) l1; da edx"
```

mscorwks – .NET ~v2.0 - 2.0.50727

clr – .NET ~v4.0 - 4.0.30319



Packer – next layer loading

next hint (AV):

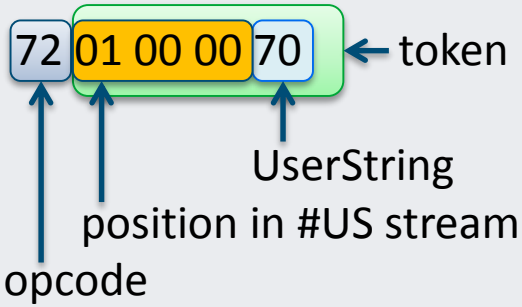
Loaded MZ is kept in memory – it can be monitored:

- New-allocated memory region
- Mapped, RW
- MZ at the begin

Probably it is executable loaded with a call to Assembly.Load() ☺

User strings

```
IL_0001: ldstr ".net testSample - console version - v1.0\n\n"
L_00000001:              72 01 00 00 70          ldstr 0x70000001
```



#US (UserString stream):

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	00	55	2E	00	6E	00	65	00	74	00	20	00	74	00	65	00	.U..n.e.t...t.e.
00000010	73	00	74	00	53	00	61	00	6D	00	70	00	6C	00	65	00	s.t.S.a.m.p.l.e.
00000020	20	00	2D	00	20	00	63	00	6F	00	6E	00	73	00	6F	00	..-...c.o.n.s.o.
00000030	6C	00	65	00	20	00	76	00	65	00	72	00	73	00	69	00	l.e...v.e.r.s.i.
00000040	6F	00	6E	00	20	00	2D	00	20	00	76	00	31	00	2E	00	o.n...-...v.1...
00000050	30	00	0A	00	0A	00	01	25	73	00	74	00	72	00	69	00	0.....%s.t.r.i.

User strings - obfuscation

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

What do the packers do?

```
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.
- Crypted strings are kept in:
  - Manifest resources
  - Static data fields
  - #US stream (crypted)

User strings - obfuscation

DecryptString methods are similar – they use:

- `!bpmd mscorlib.dll System.String.CreateStringFromEncoding`
  (confusers, eziriz, smartAssembly, cryptoobfuscator, codewall)
- `!bpmd mscorlib.dll System.String.Intern`
  (yano, babel)
- `!bpmd mscorlib.dll System.Text.StringBuilder.ToString *`
  (deepsee)

** used also by runtime*

!bpmd – (sos.dll) breakpoint for **NGEN**ed method

Ngen.exe - The Native Image Generator

User strings - obfuscation

Classic strings (*ldstr* opcode)

- `bp mscorwks!GlobalStringLiteralMap::GetStringLiteral \`
  `bp clr!StringLiteralMap::GetStringLiteral`

```
01204032   ".net testSample - console versio"
01204072   "n - v1.0.."
eax=0053d6a0 ebx=0033e42c ecx=0053d6a0 edx=0053d6ac esi=0053c940 edi=00000000
eip=792314d5 esp=0033e3b0 ebp=0033e3f0 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
mscorwks!GlobalStringLiteralMap::GetStringLiteral:
```

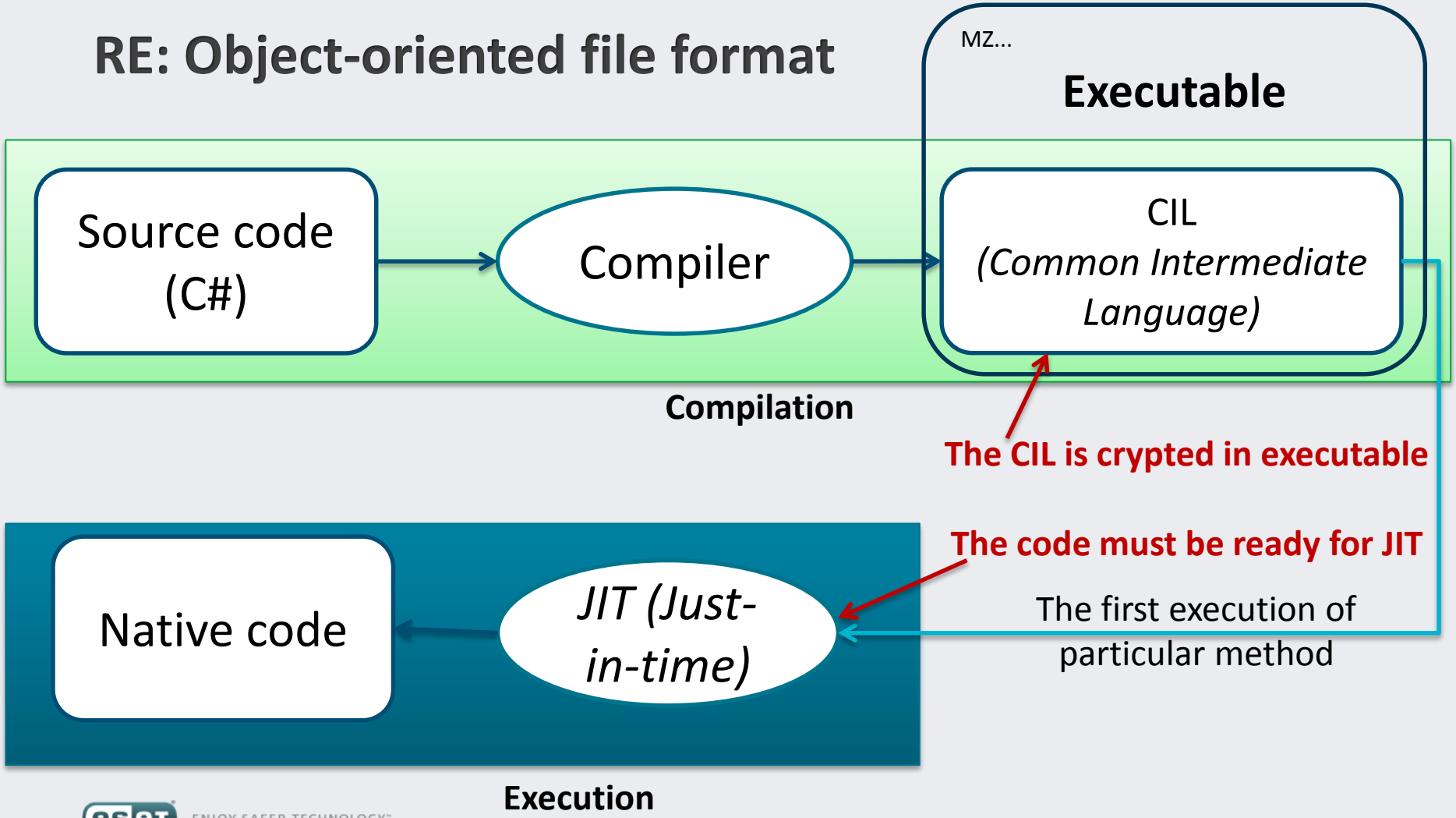
API – in CIL code

Obfuscation:

- Hide CIL code
  - Restore with Module::.cctor()
  - Restore with CompileMethod() hook

- Flow obfuscation
  - Confuse decompilation and byte patterns

RE: Object-oriented file format



API – in CIL code

Hide CIL code - solution – catching API during JIT resolving

```
bp mscorwks!MethodTable::MapMethodDeclToMethodImpl "!dumpmd dwo (esp+4)"
bp clr!MethodTable::MapMethodDeclToMethodImpl "!dumpmd ecx"
```

```
!DumpMD <Method Descriptor>
    -> Shows Method Descriptor info. We can see Code Address if method
       is jitted.
```

sos.dll

```
Method Name:  System.Math.Max(UInt32, UInt32)
Class:        721b2db4
MethodTable:  725d09cc
mdToken:      06000f22
Module:       72121000
IsJitted:     yes
CodeAddr:     72c34070
Transparency: Transparent
```

How to use it?

- Windbg scripts
  `https://bitbucket.org/marcin_hartung/vb_unpackyourtroubles`
- VB whitepaper

```
$$ runtime 4.0 - set breakpoint for display calls
.block
{
        $$ comment - load with $$><c:\wdbg\api_v4.wds

        sxe ld clr
        g
        .cordll -u -l
        .loadby sos clr

        bp clr!MethodTable::MapMethodDeclToMethodImpl "!dumpmd ecx"

}
```

How to use it?

- Next work – standalone analyser (with hooking)
  - Internal runtime methods – hook by byte patterns
  - *!bpmd* problem – NGENed functions (undocumented, complicated)
  - *!dumpmd* problem – Method Descriptor object must be parsed

References

- CFF explorer
- ILSpy
- Msdn
- .NET sources

More in the whitepaper...

Thanks!