.NET MALWARE THREAT: INTERNALS AND REVERSING

DEF CON USA 2019

by Alexandre Borges

- ✓ Malware and Security Researcher.
- ✓ Speaker at DEF CON USA 2018
- ✓ Speaker at DEF CON China 2019
- ✓ Speaker at CONFidence Conference 2019 (Poland)
- ✓ Speaker at HITB 2019 Amsterdam
- ✓ Speaker at BSIDES 2019/2018/2017/2016
- ✓ Speaker at H2HC 2016/2015
- ✓ Speaker at BHACK 2018
- ✓ Consultant, Instructor and Speaker on Malware Analysis, Memory Analysis, Digital Forensics and Rootkits.
- ✓ Reviewer member of the The Journal of Digital Forensics, Security and Law.
- ✓ Referee on Digital Investigation: The International Journal of Digital Forensics & Incident Response

Agenda:

- ❖ Introduction
- ❖ Managed executable structures
- ❖ CLR and Assembly Loader details
- ❖ .NET internals metadata
- ❖ Modules, assemblies and manifest
- ❖ .NET program structures
- ❖ Malicious code through MSIL
- ❖ .NET debugging
- ❖ Few GC and synchronous aspects
- ❖ Conclusion

✓ Last talks in conferences:

- ✓ CONFidence Conference 2019:
- ✓ https://confidence-conference.org/2019/bio.html#id=37486
- ✓ slides:
  http://www.blackstormsecurity.com/CONFIDENCE_2019_ALEXANDRE.pdf

- ✓ DEF CON China 2019:
- ✓ https://www.defcon.org/html/dc-china-1/dc-cn-1-speakers.html#Borges
- ✓ slides:
  http://www.blackstormsecurity.com/docs/DEFCON_CHINA_ALEXANDRE.pdf

- ✓ HITB Amsterdam 2019:
  https://conference.hitb.org/hitbsecconf2019ams/speakers/alexandre-borges/
- ✓ slides: http://www.blackstormsecurity.com/docs/HITB_AMS_2019.pdf

- ✓ DEF CON USA 2018:
- ✓ https://www.defcon.org/html/defcon-26/dc-26-speakers.html#Borges
- ✓ slides: http://www.blackstormsecurity.com/docs/DEFCON2018.pdf

✓ Malwoverview Tool: https://github.com/alexandreborges/malwoverview

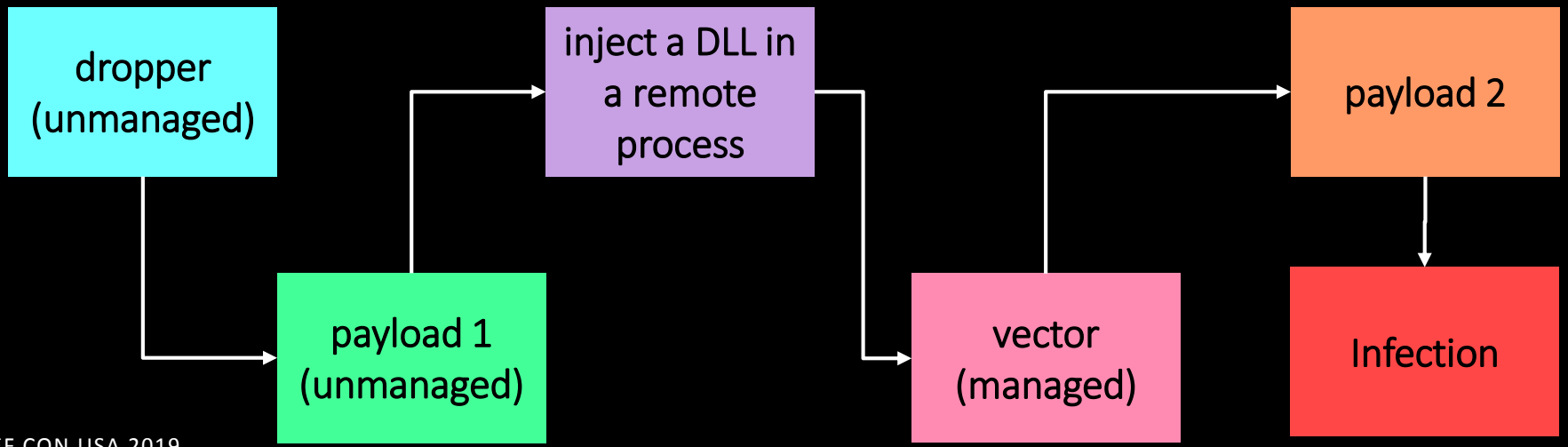
INTRODUCTION

- ✓ Motivations to this talk about .NET reversing and internals:
  - ✓ Most of the time, professionals are interested in unpacking embedded resources from a .NET sample.
  - ✓ In another moment, the concern is dumping the unpacked binary from memory.
  - ✓ Sometimes, we have looked for any unpacking routine to dynamically unpack the encrypted content.
  - ✓ All of these actions are correct and recommended.
- ✓ However….
  - ✓ Many people don't understand .NET metadata components.
  - ✓ Most people based their analysis on the decompiled code, but never on IL.
  - ✓ Malware's authors have manipulated the IL to attack the system and even the .NET runtime.

- ✓ There are many available methods to infect a system using .NET malware. Most of the time, a .NET code decrypts and loads a native code (or injects a code into a target process).

- ✓ However, there are few approaches that use indirect techniques:

  - ✓ An e-mail comes from the Internet and a first dropper is downloaded.
  - ✓ This dropper fetches a encrypted payload, which contains a native payload and a managed code.
  - ✓ The payload 1 executes and injects a DLL into a remote chosen process.
  - ✓ This DLL loads (and sometime decrypts) the malicious managed code.
  - ✓ The malicious managed code drops the payload 2 (real and advanced).
  - ✓ The true infection starts.



- ✓ It is not necessary to comment about how to inject a code because the steps are the same ever-sequence:
  - ✓ CreateToolhelp32Snapshot( ) → Module32First( ) → Module32Next( ) → comparison (wcscmp( ))
  - ✓ VirtualAllocEx( ) → WriteProcessMemory( ) → CreateRemoteThread( ) → WaitForSingleObject → VirtualFreeEx( ).
- ✓ Find the offset of injected DLL from the base module (any testing module).
- ✓ Use this offset to invoke functions from any injected remote process through GetProcessAddress( ) + CreateRemoteThread( ).
- ✓ Thi injected DLL can load the next stage and, eventually, decrypt it.
- ✓ Obviously, the .NET managed code can be loaded from any process or, even worse, from an natived injected code (DLL).
- ✓ After loading it, it is easy to execute it. Our simple case above.

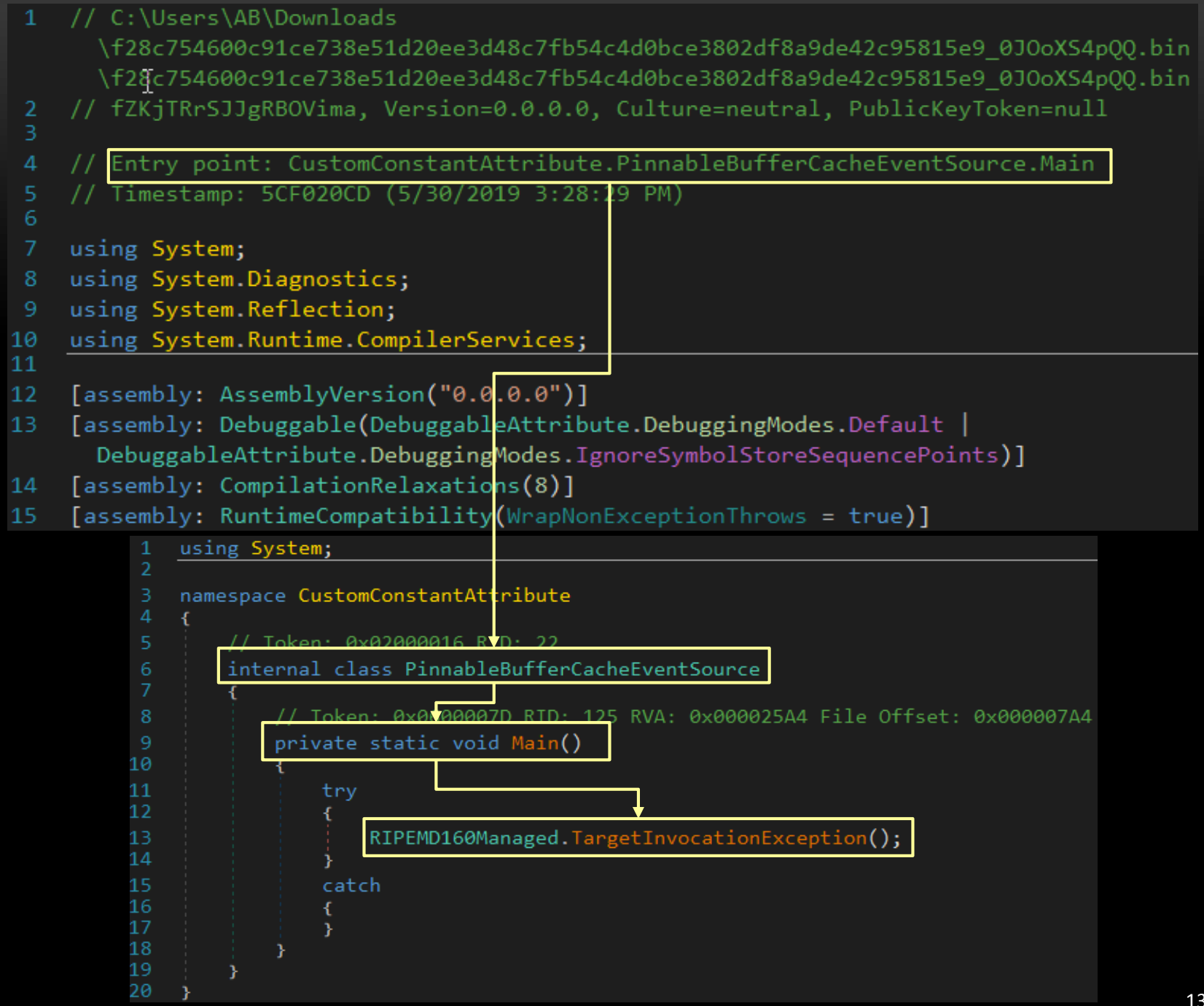
- ✓ We should remember that a typical native application can also load a .NET runtime and execute a managed code:
  - ✓ CLRCreateInstance( ): provides the ICLRMetaHost interface.
  - ✓ ICLRMetaHost::GetRunTime( ): gets the ICLRRuntimeInfo.
  - ✓ ICLRRuntimeInfo::GetInterface( ): Loads the CLR into the current process and returns runtime interface pointers.
  - ✓ ICLRRuntimeHost::ExecuteApplication( ): specifies the application to be activated in a new domain.
  - ✓ ICLRRuntimeHost::Start( ): starts the the runtime.
  - ✓ ICLRRuntimeHost::ExecuteInDefaultAppDomain( ): invokes a method in the .NET managed assembly (this steps does not work for all .NET assembly's method). Thus, in this case, starts the managed assembly. ☺
- ✓ Finally, the real infection starts. ☺

- ✓ The .NET framework is composed by:
  - ✓ CLR (Common Language Runtime), which is the .NET engine.
  - ✓ Libraries (System.IO, System.Reflection, System.Collections, ...).
- ✓ Basically:
  - ✓ source code is written in C#, F#, VB.NET and Powershell.
  - ✓ compiled to CLI (Common Language Infrastruture Code).
  - ✓ executed by the CLR.
- ✓ Tools used to reverse and analyze .NET malware threats are completely different than ones used to reverse native language:
  - ✓ dnSpy (excellent)
  - ✓ ILSpy (excellent)
  - ✓ RedGate .NET Reflector
  - ✓ De4dot (deobfuscator)
  - ✓ Microsoft Visual Studio
  - ✓ WinDbg (including SOS.dll extension)
  - ✓ DotPeek
  - ✓ IDA Pro
  - ✓ Microsoft ILASM/ILDASM (Intermediate Language Assembly/Disassembler)

- ✓ Other interesting tools to analyze and understand .NET runtime are available:
  - ✓ MemoScope.Net: https://github.com/fremag/MemoScope.Net
  - ✓ Shed -- a .NET runtime inspector: https://github.com/enkomio/shed
  - ✓ SuperDump, for automated crash dump analysis: https://github.com/Dynatrace/superdump
  - ✓ DumpMiner: https://github.com/dudikeleti/DumpMiner
  - ✓ MemAnalyzer: https://github.com/Alois-xx/MemAnalyzer
  - ✓ Sharplab: https://sharplab.io/
  - ✓ ObjectLayoutInspector to analyze internal structures of the CLR types at runtime (https://github.com/SergeyTeplyakov/ObjectLayoutInspector)
- ✓ Tools are excellent to help us, but most .NET malware threats have deployed the same tricks from native code to make our job harder: packers, obfuscation and anti-reversing techniques.
  - ✓ .NET Reactor
  - ✓ Salamander .NET Obfuscator
  - ✓ Dotfuscator
  - ✓ Smart Assembly
  - ✓ CryptoObfuscator for .NET
  - ✓ Agile
  - ✓ ArmDot
  - ✓ babelfor.NET
  - ✓ Eazfuscator.NET
  - ✓ Spice.Net
  - ✓ Skater.NET
  - ✓ VM Protect 3.40

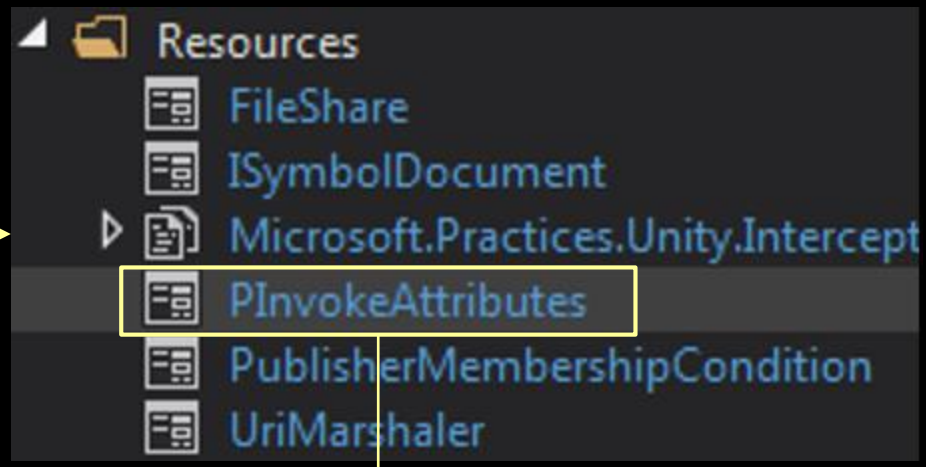
- ✓ There are many obfuscators, which perform:
  - ✓ Control flow obfuscation and dead/junk code insertion.
  - ✓ Renaming: methods signatures, fields, methods implementation, namespaces, metadata and external references.
  - ✓ Re-encoding: changing printable to unprintable characters
  - ✓ Simple encryption of methods and strings.
  - ✓ Cross reference obfuscation.
- ✓ Yes, I know... I've already talked about de-obfuscation in DEF CON China 2019. ☺
- ✓ Most time, the real and encoded malicious code (payload) is downloaded and decrypted/loaded into the memory for execution:
  - ✓ System.Reflection.Assembly.Load( )
  - ✓ System.Reflection.Assembly.LoadFile()
  - ✓ System.Reflection.MethodInfo.Invoke( )
- ✓ As we already know, Load( )/LoadFile( ) function are usually followed by:
  - ✓ GetType ( ) → GetMethod( ) → Invoke( ) (this is a typical Reflection approach)

- ✓ Another possible approach would be:
  - ✓ GetAssemblyName( ) + GetType( ) + GetMethod( ) + Invoke( )
- ✓ Some "encrypted content" is loaded from as a resource, so it is usual finding the following sequence:
  - ✓ FindResource( ) + SizeOfResource( ) + LoadResource( ) + LockResource( )
  - ✓ Resources.ResourceManager.GetObject( )
- ✓ Additionally, we've seen techniques using embedded references such as DLLs as resources through a sequence of calls using:
  - ✓ AssemblyLoader.Attach( ) + AssemblyLoader.ResolveAssembly( ).
- ✓ As you've guessed, AssemblyLoader.ResolveAssembly( ) is used to resolving assemblies that are not available at the exact time of calling other methods, which are external references to the binary itself.

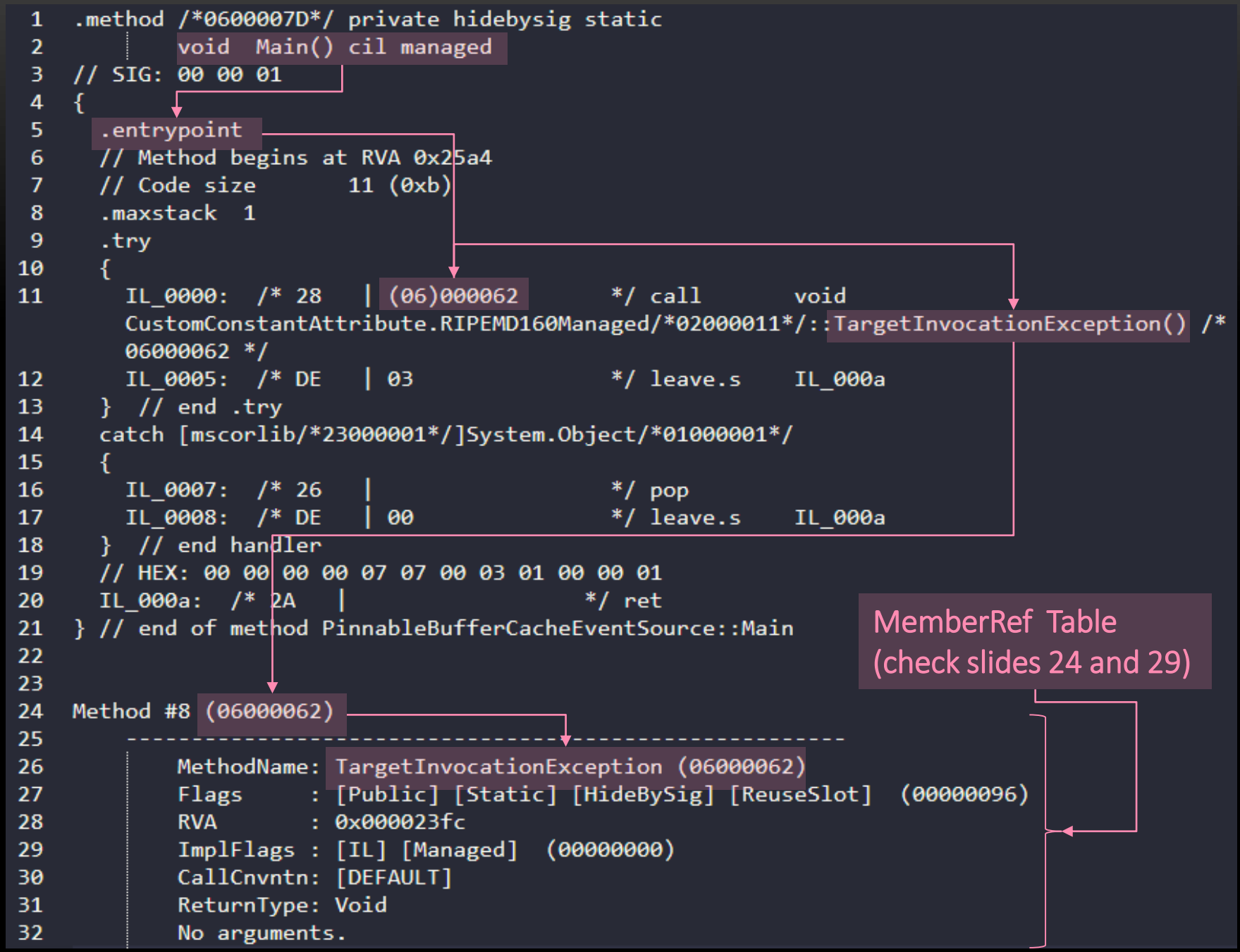

// C:\Users\AB\Downloads
\f28c754600c91ce738e51d20ee3d48c7fb54c4d0bce3802df8a9de42c95815e9_0JOoXS4pQQ.bin
// fZKjTRrSJJgRBOVima, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
// Entry point: CustomConstantAttribute.PinnableBufferCacheEventSource.Main
// Timestamp: 5CF020CD (5/30/2019 3:28:29 PM)
using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
[assembly: AssemblyVersion("0.0.0.0")]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default |
DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
using System;
namespace CustomConstantAttribute
{
// Token: 0x02000016 RID: 22
internal class PinnableBufferCacheEventSource
{
// Token: 0x0600007D RID: 125 RVA: 0x000025A4 File Offset: 0x000007A4
private static void Main()
{
try
{
RIPEMD160Managed.TargetInvocationException();
}
catch
{
}
}
}
}

```
3   public static void TargetInvocationException()
4   {
5       try
6       {
7           Assembly inputLanguageChangingEventHandler = Assembly.Load(HttpServerUtility.UrlTokenDecode
              (RIPEMD160Managed.ContentDispositionHeaderValue(string.Concat(new string[]
8           {
9               RIPEMD160Managed.PInvokeAttributesPInvokeAttributes,
10              RIPEMD160Managed.PublisherMembershipConditionPublisherMembershipCondition,
11              RIPEMD160Managed.ISymbolDocumentISymbolDocument,
12              RIPEMD160Managed.UriMarshalerUriMarshaler,
13              RIPEMD160Managed.FileShareFileShare
14          })))));
15          RIPEMD160Managed.HttpListener(inputLanguageChangingEventHandler);
16      }
17      catch
18      {
19      }
20  }
21
```

```
1   // CustomConstantAttribute.RIPEMD160Managed
2   // Token: 0x04000001 RID: 1
3   private static string
      PInvokeAttributesPInvokeAttributes =
      RIPEMD160Managed.Matrix
      (Assembly.GetExecutingAssembly(),
      "PInvokeAttributes");
4
```




```
MZ.....................@.....
.............................
..........!..L.!This program ca
nnot be run in DOS mode....$...
....PE..L.... .\..............
......p.......... ........@..
.............................
.....@.......................
........|...O.... ...l.........
```


.method /*0600007D*/ private hidebysig static
void Main() cil managed
// SIG: 00 00 01
{
.entrypoint
// Method begins at RVA 0x25a4
// Code size 11 (0xb)
.maxstack 1
.try
{
IL_0000: /* 28 | (06)000062 */ call void
CustomConstantAttribute.RIPEMD160Managed/*02000011*/::TargetInvocationException() /*
06000062 */
IL_0005: /* DE | 03 */ leave.s IL_000a
} // end .try
catch [mscorlib/*23000001*/]System.Object/*01000001*/
{
IL_0007: /* 26 | */ pop
IL_0008: /* DE | 00 */ leave.s IL_000a
} // end handler
// HEX: 00 00 00 00 07 07 00 03 01 00 00 01
IL_000a: /* 2A | */ ret
} // end of method PinnableBufferCacheEventSource::Main
Method #8 (06000062)
MethodName: TargetInvocationException (06000062)
Flags : [Public] [Static] [HideBySig] [ReuseSlot] (00000096)
RVA : 0x000023fc
ImplFlags : [IL] [Managed] (00000000)
CallCnvntn: [DEFAULT]
ReturnType: Void
No arguments.
MemberRef Table
(check slides 24 and 29)

```
.assembly fZKjTRrSJJgRBOVima

.mresource public Microsoft.Practices.Unity.InterceptionExtension.Properties.Resources.resources
{
  // Offset: 0x00000000 Length: 0x00000872
}
.mresource public PInvokeAttributes
{
  // Offset: 0x00000876 Length: 0x000006AB
}
.mresource public PublisherMembershipCondition
{
  // Offset: 0x00000F25 Length: 0x000006AB
}
.mresource public ISymbolDocument
{
  // Offset: 0x000015D4 Length: 0x000006AB
}
.mresource public UriMarshaler
{
  // Offset: 0x00001C83 Length: 0x000006AB
}
.mresource public FileShare
{
  // Offset: 0x00002332 Length: 0x00000000
}
.module fZKjTRrSJJgRBOVima
// MVID: {7FCE3D56-2F4A-474D-93C6-686282005121}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0002       // WINDOWS_GUI
.corflags 0x00000003    //  ILONLY 32BITREQUIRED
// Image base: 0x03830000
```

Manifest

- ✓ As every single malware code, this one is using Reflection to retrieve information in runtime. In this case also calls the GetExecutingAssembly( ) method to get the Assembly object, which represents the current assembly.

```
call      class [mscorlib]System.Reflection.Assembly [mscorlib]System.Reflection.Assembly::GetExecutingAssembly()
ldstr     "PInvokeAttributes"
call      string CustomConstantAttribute.RIPEMD160Managed::Matrix(class [mscorlib]System.Reflection.Assembly, string)
stsfld    string CustomConstantAttribute.RIPEMD160Managed::PInvokeAttributesPInvokeAttributes
call      class [mscorlib]System.Reflection.Assembly [mscorlib]System.Reflection.Assembly::GetExecutingAssembly()
ldstr     "PublisherMembershipCondition"
call      string CustomConstantAttribute.RIPEMD160Managed::Matrix(class [mscorlib]System.Reflection.Assembly, string)
stsfld    string CustomConstantAttribute.RIPEMD160Managed::PublisherMembershipConditionPublisherMembershipCondition
call      class [mscorlib]System.Reflection.Assembly [mscorlib]System.Reflection.Assembly::GetExecutingAssembly()
ldstr     "ISymbolDocument"
call      string CustomConstantAttribute.RIPEMD160Managed::Matrix(class [mscorlib]System.Reflection.Assembly, string)
stsfld    string CustomConstantAttribute.RIPEMD160Managed::ISymbolDocumentISymbolDocument
call      class [mscorlib]System.Reflection.Assembly [mscorlib]System.Reflection.Assembly::GetExecutingAssembly()
ldstr     "UriMarshaler"
call      string CustomConstantAttribute.RIPEMD160Managed::Matrix(class [mscorlib]System.Reflection.Assembly, string)
stsfld    string CustomConstantAttribute.RIPEMD160Managed::UriMarshalerUriMarshaler
call      class [mscorlib]System.Reflection.Assembly [mscorlib]System.Reflection.Assembly::GetExecutingAssembly()
ldstr     "FileShare"
call      string CustomConstantAttribute.RIPEMD160Managed::Matrix(class [mscorlib]System.Reflection.Assembly, string)
stsfld    string CustomConstantAttribute.RIPEMD160Managed::FileShareFileShare
ret
```

- ✓ Therefore, we can extract these resources (DLLs, for example) by using either dnSpy or ILSpy , decrypt and load them again into the managed code.
- ✓ Of course, in this case, we'll be able to see all "hidden" references, finally. ☺
- ✓ To load the "decrypted" resources into the managed code, we can use ILSpy + Reflexil plugin (http://reflexil.net/).
- ✓ Finally, it is necessary to remove the "old" references to the embedded resources (performed by AssemblyLoader.Attach( )) from the initializer (or removing the whole initializer) because, at this time, they are "decrypted".
- ✓ By the way, Reflexil is able to handle different obfuscators such as Babel NET, CodeFort, Skater NET, SmartAssembly, Spices Net, Xenocode, Eazfuscator NET, Goliath NET, ILProtector, MaxtoCode, MPRESS, Rummage, CodeVeil,CodeWall, CryptoObfuscator, DeepSea, Dotfuscator, dotNET Reactor, CliSecure and so on.
- ✓ At end, gaining knowledge in .NET internals and metadata can be interesting.

- ✓ Most time, there are module/type initializers (similar to TLS in native code) executing before classes and entry point methods.
  - ✓ .NET protectors hardly change the entry point and, usually, the trick is in the initializer.
  - ✓ .cctor( ) method is a static class constructor:
    - ✓ called before the Main( ) method (usually set as entry point), for example.
    - ✓ when the module has a .cctor (<Module>::.cctor( )), so it is run before executing any other class initializers or even an entry point.
    - ✓ It is common finding unpackers, decrypters and hooks in the .cctor( ) method.
- ✓ Hijacking the ICorJitCompiler::compileMethod( ) is an interesting and useful way to take the control of the JIT engine because this method is used to create a native code, so we find managed and native code together. ☺
- ✓ In this case: .cctor( ) → hooking compileMethod( ) → hiding/encryting user code.

.NET details

- Metadata works as descriptors for each structure component of the application: classes, attributes, members, and so on.

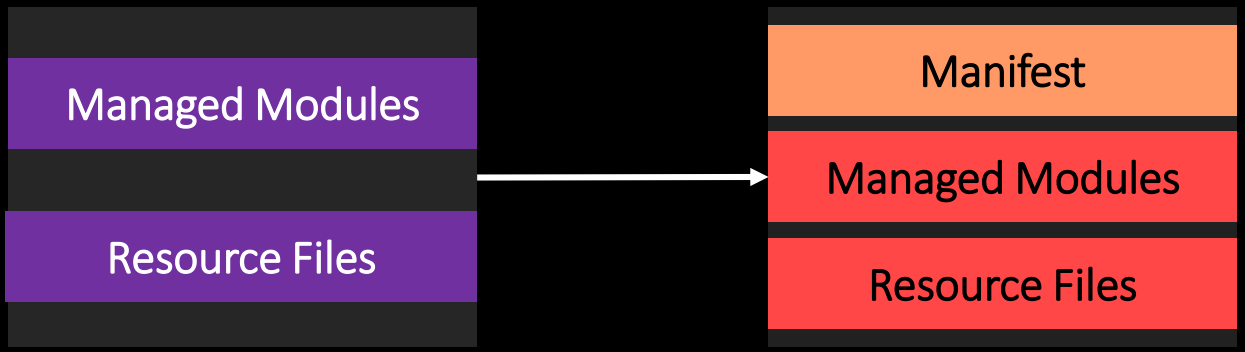
- ✓ Remember that a .NET application is composed by:
  - ✓ managed executable files, which each one contains metadata
  - ✓ managed code (optionally)

- ✓ .NET Assembly: managed .NET application (modules) + class libraries + resources files (more information later)

- ✓ CLR runtime environment: loaders + JIT compiler.

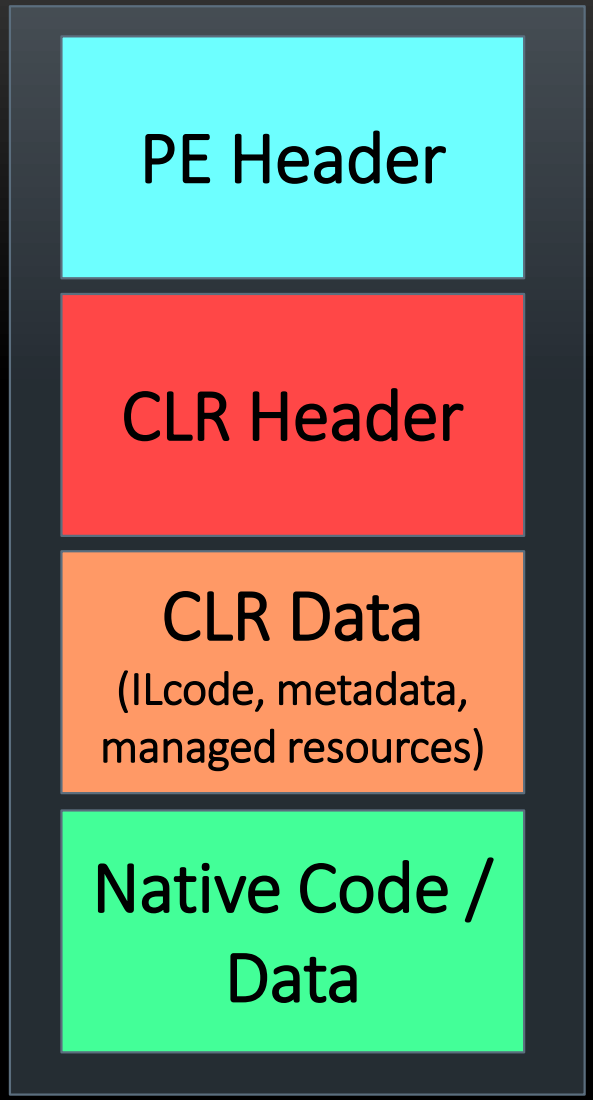
- ✓ .NET source code → .NET compiler → module (IL + metadata) → CLR (loaders + JIT compiler) → native instruction → Execution Engine

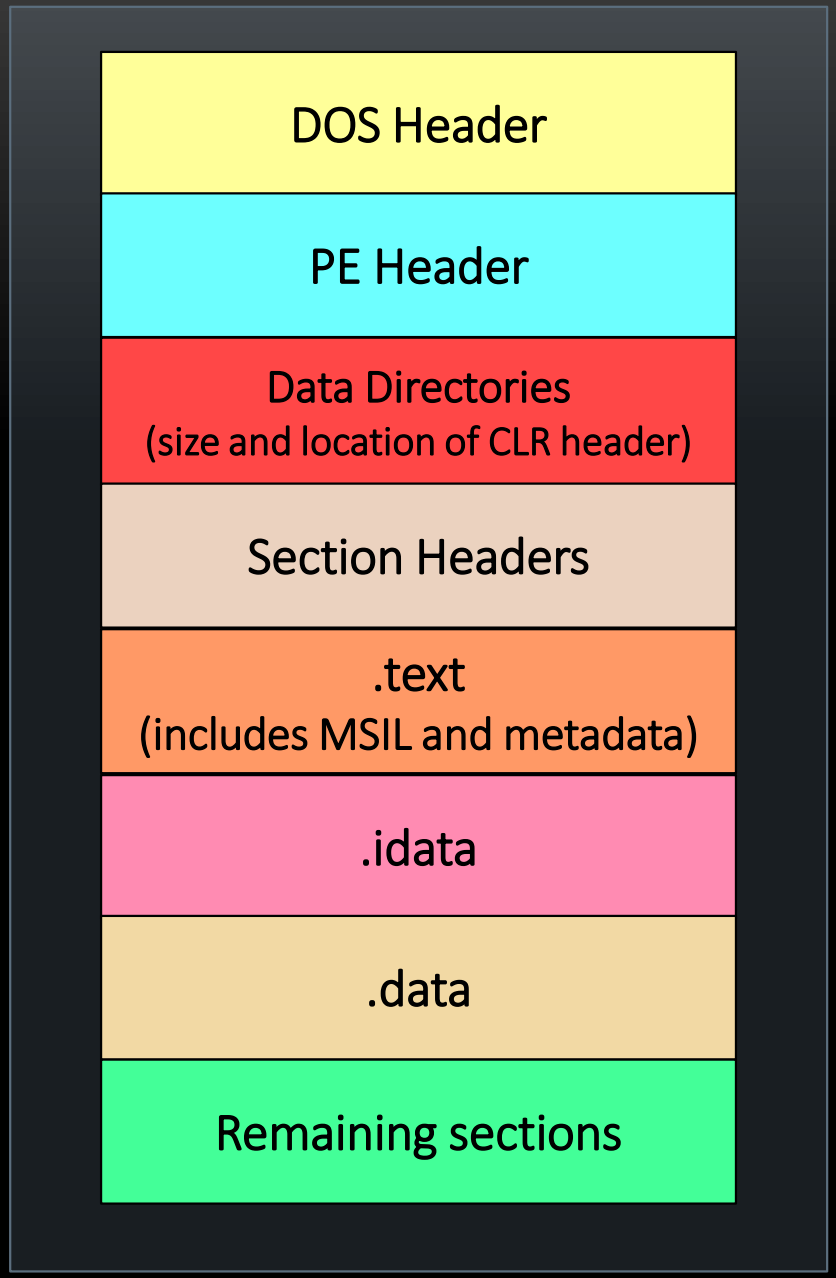
- ✓ Managed module is composed by:
  - ✓ PE header: If the module contains only IL code, so most of information of header is ignored. However, if the module also contains native code, so things are different. ☺
  - ✓ CLR header: contains the version of the CLR, token of Main( ) (natural entry poiint), resources and so on.
  - ✓ Metadata: describe types and members. Additionally, it helps the GC to track the life time of objects. ☺
  - ✓ IL (Intermediate Language) code: the managed code.



Compiler (C#, VB, F#) + Linker

.NET Assembly (.exe or .dll)





- ✓ Managed resources in contained into .text section (and not .rsrc section).

- ✓ Metadata is composed by tables such as:
  - ✓ Definition tables: ModuleDef, TypeDef, MethodDef, FieldDef, ParamDef, PropertyDef and EventDef
  - ✓ Reference tables: AssemblyRef, ModuleRef, TypeRef and MemberRef.
  - ✓ Manifest tables: AssemblyDef, FileDef, ManifestResourceDef and ExportedTypesDef.
- ✓ Most malicious .NET malware samples have:
  - ✓ Used code manipulation (encryption/decryption) in .ctor( )/.cctor( )/Finalize( )
  - ✓ Called unmanaged functions from DLLs using P/Invoke.
  - ✓ Used COM components (very usual).

```
1  ============================================================
2  ScopeName : MicroTik Realtek Driver.exe
3  MVID      : {CE979730-9E54-4211-9B33-C1D7F311A4E4}
4  ============================================================
5  Global functions
6  -------------------------------------------------------
7
8  Global fields
9  -------------------------------------------------------
10
11 Global MemberRefs
12 -------------------------------------------------------
13
14 TypeDef #1 (02000002)
15 -------------------------------------------------------
16     TypDefName: testprint.My.MyApplication  (02000002)
17     Flags     : [NotPublic] [AutoLayout] [Class] [AnsiClass]  (00000000)
18     Extends   : 01000011 [TypeRef]
       Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase
19     Method #1 (06000001) [ENTRYPOINT]
20     -------------------------------------------------------
21         MethodName: Main (06000001)
22         Flags     : [Assem] [Static] [ReuseSlot]  (00000013)
23         RVA       : 0x00002050
24         ImplFlags : [IL] [Managed] [NoInlining]  (00000048)
25         CallCnvntn: [DEFAULT]
26         ReturnType: Void
27         1 Arguments
28             Argument #1:  SZArray String
29         1 Parameters
30             (1) ParamToken : (08000001) Name : Args flags: [none] (00000000)
```

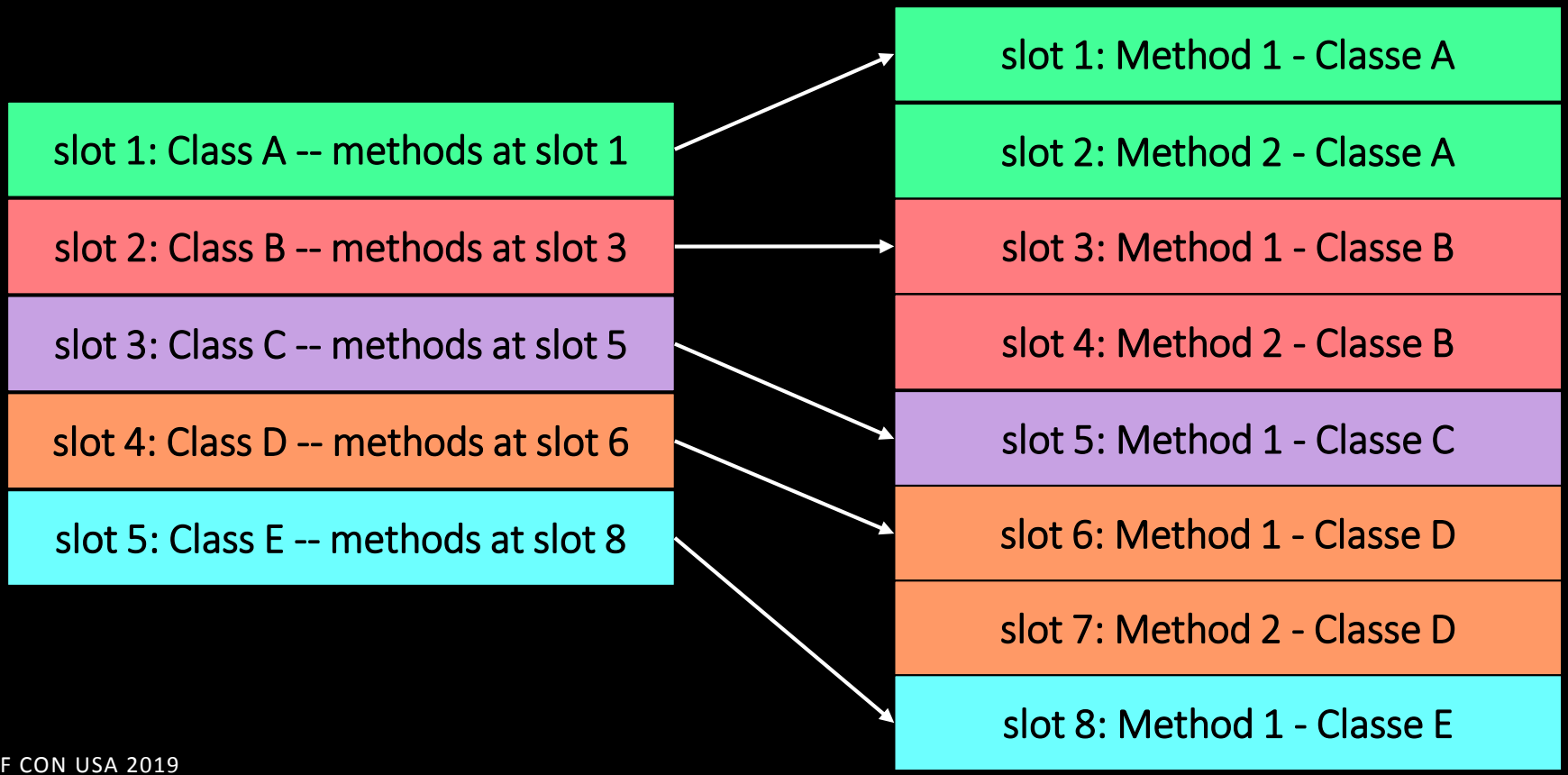
✓ ILDasm → View → MetaInfo → Show! menu:

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

```
File size                : 2467840
PE header size           : 512 (496 used)     ( 0.02%)
PE additional info       : 4643               ( 0.19%)
Num.of PE sections       : 3
CLR header size          : 72                 ( 0.00%)
CLR meta-data size       : 105828             ( 4.29%)
CLR additional info      : 2242464            (90.87%)
CLR method headers       : 3751               ( 0.15%)
Managed code             : 109504             ( 4.44%)
Data                     : 5120               ( 0.21%)
Unaccounted              : -4054              (-0.16%)

Num.of PE sections       : 3
   .text     - 2462208
   .rsrc     - 4608
   .reloc    - 512
```

✓ ILDasm.exe → View → Statistics

- ✓ Metadata describes all declared or referenced data in a module such as classes, members, attributes, properties and relationships.
- ✓ Metadata is organized as a relational database using cross-references and making possible to find what class each method comes from.
- ✓ Metadata are represented by named streams, which are classified as metadata heaps and metadata tables.



- ✓ Metadata heaps:
  - ✓ GUID heap: contains objects of size equal to 16 bytes.
  - ✓ String heap: contains strings.
  - ✓ Blog heap: contains arbitrary binary objects aligned on 4-byte boundary.
  - ✓ There can be 6 named streams:
    - ✓ #GUID: contains global unique identifiers.
    - ✓ #Strings: contains names of classes, methods, and so on.
    - ✓ #US: contains user defined strings.
    - ✓ #~: contains compressed metadata stream.
    - ✓ #-: contains uncompressed metadata stream.
    - ✓ Blob: contains metadata from binary objects.
  - ✓ An important note: compressed and uncompressed named streams are mutually exclusive.
- ✓ Metadata tables:
  - ✓ The schema defines the metadata tables by usings a descriptor.
  - ✓ There are more than 40 metadata tables.

- ✓ Tokens have 4 bytes, which the first byte determines the metadata table and the three remaining bytes are the RID.
- ✓ RID (record identifiers) are used as row indexes in metadata tables.
- ✓ Tokens determines which metadata tables are being referred.

- ✓ Unfortunately, tokens don't cover all tables (auxiliary tables, which are hardcoded).

0(0x0): Module
1(0x1): TypeRef
2(0x2): TypeDef
3(0x3): FieldPtr
4(0x4): Field
5(0x5): MethodPtr
6(0x6): Method
7(0x7): ParamPtr
8(0x8): Param
9(0x9): InterfaceImpl
10(0xa): MemberRef
11(0xb): Constant
12(0xc): CustomAttribute
13(0xd): FieldMarshal
14(0xe): DeclSecurity
15(0xf): ClassLayout
16(0x10): FieldLayout
17(0x11): StandAloneSig
18(0x12): EventMap
19(0x13): EventPtr
20(0x14): Event
21(0x15): PropertyMap
22(0x16): PropertyPtr
23(0x17): Property
24(0x18): MethodSemantics
25(0x19): MethodImpl
26(0x1a): ModuleRef
27(0x1b): TypeSpec
28(0x1c): ImplMap
29(0x1d): FieldRVA
30(0x1e): ENCLog
31(0x1f): ENCMap
32(0x20): Assembly
33(0x21): AssemblyProcessor
34(0x22): AssemblyOS
35(0x23): AssemblyRef
36(0x24): AssemblyRefProcessor
37(0x25): AssemblyRefOS
38(0x26): File
39(0x27): ExportedType
40(0x28): ManifestResource
41(0x29): NestedClass
42(0x2a): GenericParam
43(0x2b): MethodSpec
44(0x2c): GenericParamConstraint

```
CLR meta-data size  : 105828
   Module          -     1 (10 bytes)
   TypeDef         -    36 (504 bytes)     0 interfaces, 0 explicit layout
   TypeRef         -   215 (1290 bytes)
   MethodDef       - 1049 (14686 bytes)   0 abstract, 0 native, 951 bodies
   FieldDef        -   461 (2766 bytes)    6 constant
   MemberRef       -   447 (2682 bytes)
   ParamDef        -   789 (4734 bytes)
   Constant        -    20 (120 bytes)
   CustomAttribute- 1479 (8874 bytes)
   NativeType      -    14 (56 bytes)
   ClassLayout     -     1 (8 bytes)
   StandAloneSig   -    58 (116 bytes)
   PropertyMap     -     9 (36 bytes)
   Property        -   404 (2424 bytes)
   MethodSemantic-   756 (4536 bytes)
   TypeSpec        -     9 (18 bytes)
   ModuleRef       -     8 (16 bytes)
   Assembly        -     1 (22 bytes)
   AssemblyRef     -     6 (120 bytes)
   ManifestResource-     4 (48 bytes)
   NestedClass     -    19 (76 bytes)
   EventMap        -     2 (8 bytes)
   Event           -     3 (18 bytes)
   ImplMap         -    54 (432 bytes)
   FieldRVA        -     2 (12 bytes)
   GenericParam    -     5 (40 bytes)
   MethodSpec      -    12 (48 bytes)
   GenericParamConstraint-    2 (8 bytes)
   Strings         - 23400 bytes
   Blobs           -  9716 bytes
   UserStrings     - 28740 bytes
   Guids           -    16 bytes
   Uncategorized -   248 bytes
```

- ✓ ILDasm.exe → View → Statistics

```
String Heap:  23400(0x5b68) bytes
00000001: _Closure$__101-0
00000012: get_Label10
0000001e: set_Label10
0000002a: get_PictureBox10
0000003b: set_PictureBox10
0000004c: get_GroupBox10
0000005b: set_GroupBox10
0000006a: get_Label20
00000076: set_Label20
00000082: get_PictureBox20
00000093: set_PictureBox20
000000a4: get_GroupBox20
000000b3: set_GroupBox20
....

User Strings
-----------------------------------------------------
70000001 : (19) L"testprint.Resources"
70000029 : (11) L"base chile2"
70000041 : (11) L"BC_benlinea"
70000059 : (16) L"bc_btc_confirmar"
7000007b : (16) L"bc_btn_ingressar"
7000009d : (13) L"bc_clava_text"
700000b9 : (11) L"bc_digipass"
700000d1 : (14) L"bc_ingresa_sms"
700000ef : (19) L"bc_maxima_seguridad"
70000117 : (11) L"bc_sms_text"
7000012f : ( 7) L"bc_wait"
7000013f : ( 8) L"be_ahora"
70000151 : (14) L"be_btc_engresa"
...
```

- ✓ IDAasm → View → MetaInfo → RawHeap
- ✓ ILDasm → View → MetaInfo → Show!

- ✓ Check the installed .NET version:
  - ✓ Subdirectories under C:\Windows\Microsoft.NET
  - ✓ clrver.exe
  - ✓ clrver.exe -all
- ✓ Programming directly in IL (Intermediate Language) can be interesting because:
  - ✓ IL is stack based, so we don't find any instruction related to register manipulation. ☺
  - ➢ Ngen.exe can be used to compile IL instructions to native code.
  - ➢ Eventually, malware threats have attacked the .NET runtime to subvert the system. ☺
- ✓ Assemblies can be classified as:
  - ✓ private: it is specific of an application and deployed at same directory.
  - ✓ shared: it is shared and used by other applications.

- ✓ In .NET applications:
  - ✓ .NET Assembly:
    - ✓ In malware samples, we usually find that resources are encrypted binaries and DLLs. ☺
    - ✓ Remember that the application can download assembly files from a URL (codeBase element).
    - ✓ .NET malware have used multi-file assemblies, partitioning types over different files. Unfortunately, it is only possible to create multfile assembly in the command line. ☺
    - ✓ Few malware authors have create .NET malware containing different types: such as C# and VB in the same assembly.

- ✓ Compile multi-file .NET malware is pretty easy:
  - ✓ csc.exe /t:module hooking.cs
  - ✓ csc.exe /t:module injection.cs
  - ✓ csc.exe /out:malwarelib.dll /t:library /addmodule:hooking.netmodule /addmodule:injection.netmodule Defcon.cs
- ✓ In this case, we have a multi-file assembly:
  - ✓ includes a managed module named hooking.netmodule and injection.netmodule. The output file is a DLL named malwarelib.dll
  - ✓ a manifest file wrapping everything.
- ✓ This compiling command add the hooking.mod file to the FileDef manifest metadata table and the its exported types to the ExportedTypeDef manifest metadata table.
- ✓ To check: ILDasm → View → MetaInfo → Show! and look for the FileDef and ExportedTypeDef tables.

Manifest

```
// Metadata version: v2.0.50727
.module extern gdi32
.module extern user32.dll
.module extern user32
.module extern gdi32.dll
.module extern Gdi32.dll
.module extern kernel32.dll
.module extern User32.dll
.module extern dwmapi.dll
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
}
.assembly extern System
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
}
.assembly extern Microsoft.VisualBasic
{
  .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )
  .ver 8:0:0:0
}
```

Native modules referred by the assembly. The module name is in the **ModuleRef.**

External assemblies that referred by the assembly **(AssemblyRef table).**

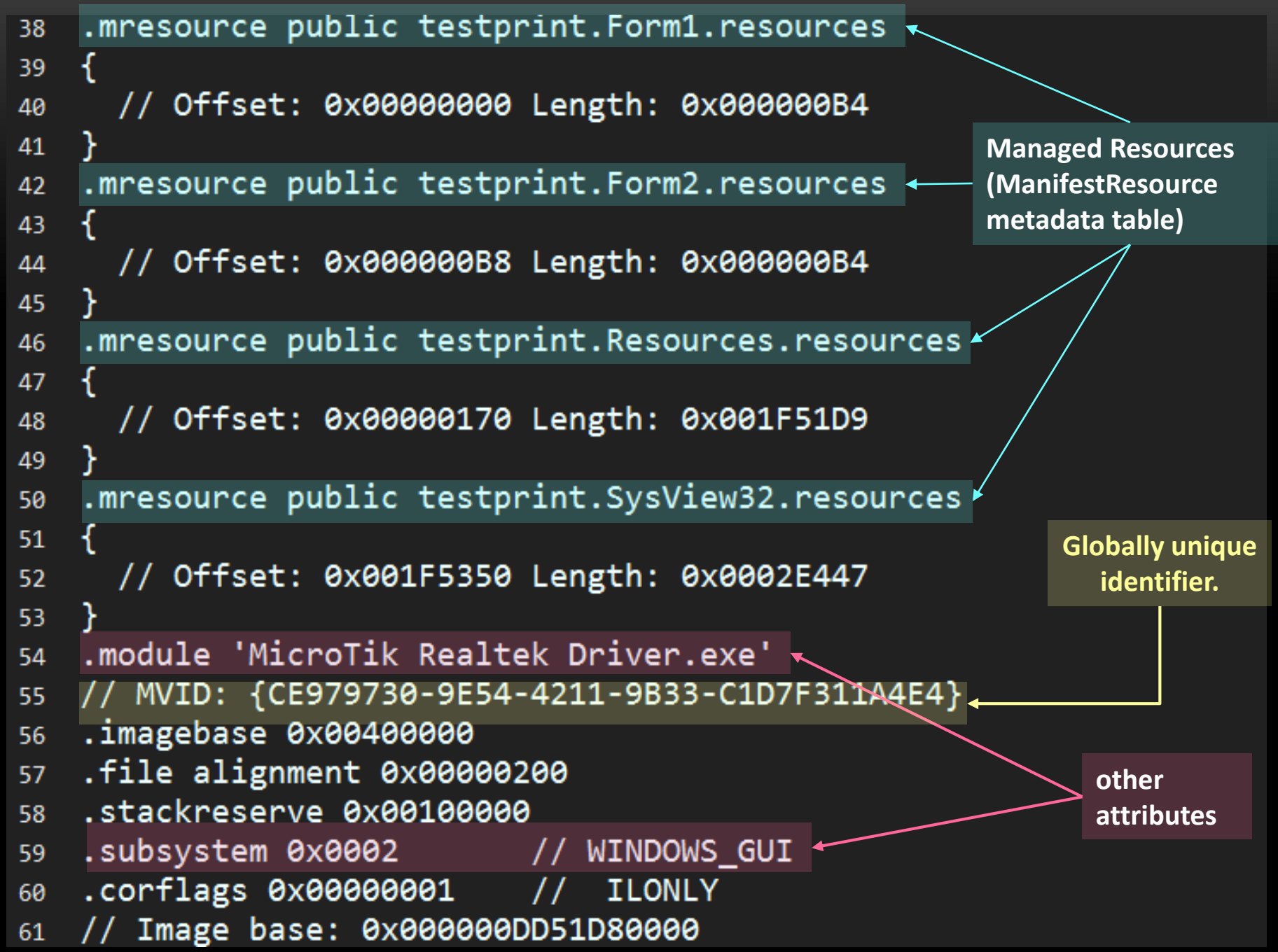
Used when a strong assembly is specified.

```
26  .assembly 'MicroTik Realtek Driver'
27  {
28    .custom instance void
      [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAt
      tribute::.ctor(int32) = ...
29    .custom instance void
      [mscorlib]System.Runtime.CompilerServices.RuntimeCompatibilityAttr
      ibute::.ctor() = ...
30
31    // --- The following custom attribute is added automatically, do
      not uncomment -------
32    //  .custom instance void
      [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(valuetype
      [mscorlib]System.Diagnostics.DebuggableAttribute/DebuggingModes)
      = ...
33    .custom instance void
      [mscorlib]System.Reflection.AssemblyTitleAttribute::.ctor(string)
      = ...
34    ...
35    .hash algorithm 0x00008004
36    .ver 5:7:6:5
37  }
```

Assembly name

Custom attributes used by the compiler (or tools) and defined in the CustomAttribute metadata table (0x0C).

CALG_SHA1


.mresource public testprint.Form1.resources
{
// Offset: 0x00000000 Length: 0x000000B4
}
.mresource public testprint.Form2.resources
{
// Offset: 0x000000B8 Length: 0x000000B4
}
.mresource public testprint.Resources.resources
{
// Offset: 0x00000170 Length: 0x001F51D9
}
.mresource public testprint.SysView32.resources
{
// Offset: 0x001F5350 Length: 0x0002E447
}
.module 'MicroTik Realtek Driver.exe'
// MVID: {CE979730-9E54-4211-9B33-C1D7F311A4E4}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0002 // WINDOWS_GUI
.corflags 0x00000001 // ILONLY
// Image base: 0x000000DD51D80000
Managed Resources (ManifestResource metadata table)
Globally unique identifier.
other attributes

- ✓ Of course, we could have a “big malware module” to use in projects:
  - ✓ al.exe /out: BigMalwareLib.dll /t:library hooking. netmodule injection. netmodule
  - ✓ csc.exe /t:module /r:BigMalwareLib.dll Defcon.cs
  - ✓ al /out:Defcon.exe /t:exe /main:Defcon.Main Defcon.netmodule
- ✓ In this case, the __EntryPoint() global function will contain the Defcon::Main( ) function call (check the IL code to confirm it).
- ✓ It is not necessary to mention that malware’s authors usually don’t write strong assemblies, which as signed with the private/public key pair from the publisher. Unless that this key pair has been stolen... ☺
  - ✓ csc.exe /out:TestProgram.exe /t:exe Program.cs
  - ✓ sn.exe -k AlexandreBorges.snk
  - ✓ sn.exe -p AlexandreBorges.snk AlexandreBorges.PublicKey sha256
  - ✓ Sn.exe -tp AlexandreBorges.PublicKey
  - ✓ csc.exe /out:TestProgram.exe /t:exe /keyfile:AlexandreBorges.snk Program.cs

```
Assembly
-------------------------------------------------------
    Token: 0x20000001
    Name : Program
    Public Key    : 00 24 00 00 04 80 00 00  94 00 00 00 06 02 00 00  00 24 00 00 52 53 41 31
                  : 00 04 00 00 01 00 01 00  a3 a1 6e bc 22 0e e0 dd  db 2b bf ec 00 4c 14 c0
                  : 71 d6 11 30 70 62 28 c0  cf 77 46 95 a1 82 fc 83  90 20 4a e5 df 2e 0f 41
                  : 5d c8 d0 fc df fc 5b bc  fe 94 3c 4e 66 2f c5 65  b4 6b 8c d0 15 9c 09 19
                  : 3e 9c f8 3d 63 d7 51 ef  fd ef ba be a2 d8 47 e8  30 ba 57 b1 ee 93 33 2d
                  : 89 70 60 51 78 51 2d 0a  65 4b fb 16 5f 24 a4 f8  bb 98 0a 6d ec f1 74 58
                  : fd 89 4b 94 d2 7d 4a 7e  da 91 6b b1 89 30 02 ac
    Hash Algorithm : 0x00008004
    Version: 0.0.0.0
    Major Version: 0x00000000
    Minor Version: 0x00000000
    Build Number: 0x00000000
    Revision Number: 0x00000000
    Locale: <null>
    Flags : [PublicKey]  (00000001)
    CustomAttribute #1 (0c000001)
    -----------------------------------------------------
        CustomAttribute Type: 0a000001
        CustomAttributeName: System.Runtime.CompilerServices.CompilationRelaxationsAttribute ::
        instance void .ctor(int32)
        Length: 8
        Value : 01 00 08 00 00 00 00 00                         >                <
        ctor args: (8)

    CustomAttribute #2 (0c000002)
    -----------------------------------------------------
        CustomAttribute Type: 0a000002
        CustomAttributeName: System.Runtime.CompilerServices.RuntimeCompatibilityAttribute ::
        instance void .ctor()
        Length: 30
        Value : 01 00 01 00 54 02 16 57  72 61 70 4e 6f 6e 45 78 >    T  WrapNonEx<
                      : 63 65 70 74 69 6f 6e 54  68 72 6f 77 73 01       >ceptionThrows   <
        ctor args: ()
```

Public key generated can be viewed by:

sn.exe -p AlexandreBorges.snk AlexandreBorges.PublicKey sha256

- ✓ Once the system is compromised through a native malware and we have access to the system as administrator, so it is possible to copy our .NET assembly to the Global Assembly Cache (GAC). The Registry is not changed.
- ✓ Once a malicious .NET assembly (first stage, as a resource library) is copied to GAC, so it can be accessed by other assemblies.
- ✓ Thus, other malicious .NET malware samples (second stage) can access methods and types from the first stage.
- ✓ Only strong assemblies (signed) can be copied to the GAC (located at C:\Windows\Microsoft.NET\assembly) by using GACUtil.exe /i command.
- ✓ Futhermore, including /r option integrates the assembly with the Windows install engine.
- ✓ Unfortunately, the GACUtil.exe is not available in home-user systems, but it is possible to use the MSI to install the malware threat into the GAC. ☺
- ✓ At end, it is still feasible to using delay signing, which is a partial signing only using the public key. Therefore, private key is not used (and there isn't real protection).

- ✓ The delay signing allows that the malicious assembly to be installed into the GAC and, worse, other assemblies can make reference to it. ☺

  - ✓ csc.exe /out:malware.dll /t:exe Program.cs
  - ✓ sn.exe -k AlexandreBorges.snk
  - ✓ sn.exe -p AlexandreBorges.snk AlexandreBorges.PublicKey sha256
  - ✓ sn.exe -tp AlexandreBorges.PublicKey
  - ✓ csc.exe /out:malware.dll /t:exe /keyfile:AlexandreBorges.PublicKey /delaysign Program.cs
  - ✓ sn.exe -Vr malware.dll (CLR trust in the assembly without using the hash).

- ✓ Using csc.exe /resource makes simple to add resources (generated by resgen.exe , for example). It updates the the ManifestResourceDef table.

- ✓ It is not so hard to perform a supply-chain attack because, when a file is specified as reference in the csc.exe compiler using /reference switch, it looks at:

  - ✓ the working directory
  - ✓ csc.exe directory
  - ✓ directory specified by the /lib switch
  - ✓ directory specified by the LIB environment variable.

- ✓ Several malware samples have been modified or written directly in ILAsm to bypass common tools.

- ✓ While ILAsm is not complicated, maybe it is still recommended to remember few directives and instructions. ☺

- ✓ .assembly DefCon { }: identifies the current assembly as being DefCon.
- ✓ .assembly extern <assemblyname>: determines the external managed assembly used by the program. For example, .assembly extern <mscorlib>
- ✓ .module malware.dll: identifies the current module.
- ✓ .namespace Conference: identities the namespace, but it does not represent a metadata.
- ✓ .class public auto ansi Hacker entends [mscorlib]System.Object. Its keywords;

  - ✓ .class: identifies the current class (Hacker)
  - ✓ public: specifies the visibility. For example, it could be “private”.
  - ✓ auto: determines the class layout style. It could be “explicit” and “sequencial”.
  - ✓ ansi: string encode while communicating to unmaged code. Other values are autochar and unicode.
  - ✓ extends: determines its base class

- ✓ Other flags for .class directive are:
  - ✓ private: used with private classes, which are not visible outside the current assembly.
  - ✓ sealed: the current class can't be derived from this class.
  - ✓ abstract: the current class can't be instantiated (it holds abstract methods).
  - ✓ explicit: the loader preserve the order of fields in the memory.
  - ✓ sequential: the loader preserves the order of the instance fields as specified in the class.
  - ✓ nested family: the class is visible from the descendants of the current class only.
  - ✓ nested assembly: the class is visible only from the current assembly.
  - ✓ nested famandassem: the class is visible from the descendants of the current class, but residing in the same assembly only.
  - ✓ windowsruntime: the class is a Windows runtime type.
- ✓ .class public enum Exam: declares a class enumeration named "Exam".
- ✓ .ctor( ): instance constructor, which is related to instance fields.
- ✓ .cctor( ): class constructor (known as type initializer), which is related to static fields.

- ✓ During malware analysis, take care: there can be a global .cctor directive, which it is related to global fields.

- ✓ call: call a method. Its possible keywords:

  - ✓ return type: void, int32, and so on.
  - ✓ vararg: variable number of arguments

- ✓ calli: directive used to call methods indirectly by taking arguments + function pointer.

  - ✓ ldc.i4.0
  - ✓ ldc.i4.1
  - ✓ ldc.i4.2
  - ✓ ldftn void DefCon::Test(int32, int32, int32)
  - ✓ calli void(int32, int32, int32)

- ✓ (method reference):

  - ✓ call instance void DefCon::Exam(int32, int32, int32)
  - ✓ call instance [.module malware.dll]::Hooking(int32, int32, native int)

- ✓ .field: specifies a variable of any type declared directly in the class (or struct). Its main keywords can be:
  - ✓ public / assembly / family (accessed by any decending class) / private
  - ✓ static (shared by all instances of the referred class).
- ✓ .method: specifies the method declaration. Its main keywords (flags) can be:
  - ✓ public / static: similar meaning as especified in "field" explanation above. ☺
  - ✓ cil managed: it means this method is represented in managed code.
  - ✓ newslot: creates a new slot in the virtual table of the class to prevent that a existing method (same name and signature) to be overriden in a derived class.
  - ✓ native unmanaged: it means this method is represented in a native code.
  - ✓ abstract: of course, no implementation is provided.
  - ✓ final: as known, the method can't be overridden.
  - ✓ virtual: method can be "redefined" in derived classes.
  - ✓ strict: this method can only be overridden whether it is accessible from the class that is overriding it. Of course, the method must be virtual.
  - ✓ noinline: it is not allowed to replace calls to this method by an inline version.
  - ✓ pinvokeimpl: declares an unmanaged method from a managed code (it's is also known as P/Invoke mechanism).

- ✓ `.method public hidebysig static pinvokeimpl("user32.dll" winapi) int32 FindWindow(string,string) cil managed preservesig`

  - ✓ preservesig: return of method must be preserved.
  - ✓ FindWindows(string,string): function invoked from the "user32.dll" and that returns a int32 value.

- ✓
```
.class public DefCon implements InterfaceA,InterfaceB {
    .method void virtual int32 IfB_Speaker(string) {
      .override InterfaceB::Speaker
...
}
```

- ✓
```
.class public DefConChina extends DefCon {
    .method public specialname void .ctor( ) {
      ldarg.0
      call instance void DefCon::.ctor( )
      ret }

  callvirt instance void DefCon::IfB_Speaker( )
```

- ✓ .entrypoint: identifies the method as the entry point of the assembly.
- ✓ .maxstack: defines the maximum stack depth used by the function code

- ✓ .locals int: defines the local variable of the current method and the "init" keyword is initializing the variable with "zero" (for example, a integer variable).
- ✓ .data <var_1>: defines a data segment named "var_1".

- ✓ stloc <var>: retrieves the value returned by the call and stores into the "var" variable.
- ✓ ldarg.0: Load argument 0 onto the stack.

- ✓ ldloc <var>: copies the value of "var" onto the stack. Variants, after optmization and run, such as ldloc.0, ldloc.1, ldloc2 and ldloc3 (representing the first local variables) are possible.
- ✓ ldstr: loads the reference to a string onto stack.

- ✓ ldsflda: loads the reference of a static field onto the stack.
- ✓ ldsfld: loads the value of a static field onto the stack.
- ✓ ldc.i4 8: loads the constant value 8 onto the stack.

- ✓ br Borges: its unconditional jump similar to “jmp” in native assembly. In this case, jumping to “Borges” label.
- ✓ brtrue DefCon: takes an item from stack and, if it is zero, so jumps to “Alex” branch. Similar to jz instruction.
- ✓ brfalse Alex: takes an item from stack and, if it is one, so jumps to “Alex” branch. Similar to jnz instruction.
- ✓ .this: it is a reference to the current class (not instance of the class like C++).
- ✓ .base: it is a reference to the parent of the current class.
- ✓ .typedef: creates a alias to a type.
- ✓ .try / catch: the same meaning of traditional C language.

```
1  .assembly 'MicroTik Realtek Driver'
2  {
3    .custom instance void
     [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.ctor
     (int32) = ...
4    .custom instance void
     [mscorlib]System.Runtime.CompilerServices.RuntimeCompatibilityAttribute::.ctor()
      = ...
5  }
6
7  .module 'MicroTik Realtek Driver.exe'
8  // MVID: {CE979730-9E54-4211-9B33-C1D7F311A4E4}
9  .imagebase 0x00400000
10 .file alignment 0x00000200
11 .stackreserve 0x00100000
12 .subsystem 0x0002       // WINDOWS_GUI
13 .corflags 0x00000001    //  ILONLY
```

```
.class public auto ansi testprint.Client
    extends [mscorlib]System.Object
{
    // Nested Types
    .class nested public auto ansi sealed NeuerTextIstDaEventHandler
        extends [mscorlib]System.MulticastDelegate
    {
        // Methods
        .method /* 060003EB */ public specialname rtspecialname
            instance void .ctor (
                object TargetObject,
                native int TargetMethod
            ) runtime managed
        ....

    // Fields
    .field private class [System]System.Net.Sockets.TcpClient TcpClient
    .field private class [mscorlib]System.IO.StreamReader Str
    .field private class [mscorlib]System.IO.StreamWriter Stw
    .field private string IP
    .field private int32 Port
    .field private bool Verbunden
    .field public string casa
    .field public string Active_windows
    .field private class testprint.Client/NeuerTextIstDaEventHandler
    NeuerTextIstDaEvent
```

auto: loader defines the "best lay out" in the memory"

nested and sealed class!

specialname flag helps the loader to understand this is a special function (constructor)

Remember that a **field** is a variable of any type that is declared directly in a class or struct.

reserving 8 slots for arguments.

```
30      // Methods
31      .method /* 06000044 */ public specialname rtspecialname
32          instance void .ctor () cil managed
33      {
34          // Method begins at RVA 0x282b
35          // Code size 25 (0x19)
36          .maxstack 8
37
38          IL_0000: ldarg.0
39          IL_0001: call instance void [mscorlib]System.Object::.ctor() /* 0A000037 */
40          IL_0006: ldarg.0
41          IL_0007: ldc.i4.0
42          IL_0008: stfld bool testprint.Client::Verbunden /* 04000010 */
43          IL_000d: ldarg.0
44          IL_000e: ldstr "" /* 700004C7 */
45          IL_0013: stfld string testprint.Client::Active_windows /* 04000012 */
46          IL_0018: ret
47      } // end of method Client::.ctor
48
49      .method /* 06000045 */ public specialname
50          instance void add_NeuerTextIstDa (
51              class testprint.Client/NeuerTextIstDaEventHandler obj
52          ) cil managed
53      {
54          .custom instance void
            [mscorlib]System.Runtime.CompilerServices.CompilerGeneratedAttribute::.ctor() = (
55              01 00 00 00
56          )
```

instance constructor

load argument 0 onto the stack.

push 0 onto stack as int32.

replaces the value of a field with a value from stack.

loads a string reference onto stack.

Using a custom attribute statement to set the value of CompilerGeneratedAttribute .

```
1  .class private auto ansi sealed testprint.libera
2      extends [mscorlib]System.Object
3  {
4      .custom instance void
       [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.StandardModuleAttribute::
       .ctor() = (
5          01 00 00 00
6      )
7      // Methods
8      .method /* 060000E8 */ private static pinvokeimpl("kernel32.dll" autochar lasterr
       winapi)
9          bool SetProcessWorkingSetSize (
10             native int procHandle,
11             int32 min,
12             int32 max
13         ) cil managed preservesig
14     {
15     } // end of method libera::SetProcessWorkingSetSize
16
17     .method /* 060000E9 */ public static
18         void LiberarMemoria () cil managed
19     {
20         // Method begins at RVA 0xacc0
21         // Code size 35 (0x23)
22         .maxstack 3
23         .locals /* 11000003 */ init (
24             [0] class [mscorlib]System.Exception
25         )
26
```

Declares a private class.

Invoking an unmanaged methods.

Defines an initially runtime zeroed local variable (type class) of the current method.

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

Calls a static method named GetCurrentProcess( ) from Process class (within namespace System.Diagnostics) and returning an instance of Process class. ☺

It calls a virtual method, which can be overriden the the derived class.

```
27        .try
28        {
29            IL_0000: call class [System]System.Diagnostics.Process
              [System]System.Diagnostics.Process::GetCurrentProcess() /* 0A000113 */
30            IL_0005: callvirt instance native int
              [System]System.Diagnostics.Process::get_Handle() /* 0A00015A */
31            IL_000a: ldc.i4.m1
32            IL_000b: ldc.i4.m1
33            IL_000c: call bool testprint.libera::SetProcessWorkingSetSize(native int,
              int32, int32) /* 060000E8 */
34            IL_0011: pop
35            IL_0012: leave.s IL_0022
36        } // end .try
37        catch [mscorlib]System.Exception
38        {
39            IL_0014: dup
40            IL_0015: call void
              [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.ProjectData::
              SetProjectError(class [mscorlib]System.Exception) /* 0A000052 */
41            IL_001a: stloc.0
42            IL_001b: call void
              [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.ProjectData::
              ClearProjectError() /* 0A000053 */
43            IL_0020: leave.s IL_0022
44        } // end handler
```

It loads -1 onto the stack.

legal instructions to way out of a “try block.”

Duplicate the value of the top of the stack.

```
148     // Fields
149     .field private static class testprint.KeyboardHook/KeyDownEventHandler KeyDownEvent
150     .custom instance void
        [mscorlib]System.Runtime.CompilerServices.CompilerGeneratedAttribute::.ctor() = (
151         01 00 00 00
152     )
153     .field private static class testprint.KeyboardHook/KeyUpEventHandler KeyUpEvent
154     .custom instance void
        [mscorlib]System.Runtime.CompilerServices.CompilerGeneratedAttribute::.ctor() = (
155         01 00 00 00
156     )
157     .field private static literal int32 WH_KEYBOARD_LL = int32(13)
158     .field private static literal int32 HC_ACTION = int32(0)
159     .field private static literal int32 WM_KEYDOWN = int32(256)
160     .field private static literal int32 WM_KEYUP = int32(257)
161     .field private static literal int32 WM_SYSKEYDOWN = int32(260)
162     .field private static literal int32 WM_SYSKEYUP = int32(261)
163     .field private class testprint.KeyboardHook/KBDLLHookProc KBDLLHookProcDelegate
164     .field private native int HHookID
165
166     // Methods
167     .method /* 060000DE */ private hidebysig static pinvokeimpl("User32.dll" autochar stdcall)
168         int32 SetWindowsHookEx (
169             int32 idHook,
170             class testprint.KeyboardHook/KBDLLHookProc HookProc,
171             native int hInstance,
172             int32 wParam
173         ) cil managed preservesig
174     {
175     } // end of method KeyboardHook::SetWindowsHookEx
```

It seems that someone is interested in our typed information. ☺

```
509     .method /* 060000E7 */ family hidebysig strict virtual
510         instance void Finalize () cil managed
511     {
512         // Method begins at RVA 0xac94
513         // Code size 42 (0x2a)
514         .maxstack 8
515
516         IL_0000: ldarg.0
517         IL_0001: ldfld native int testprint.KeyboardHook::HHookID /* 04000055 */
518         IL_0006: ldsfld native int [mscorlib]System.IntPtr::Zero /* 0A0000EF */
519         IL_000b: call bool [mscorlib]System.IntPtr::op_Equality(native int, native int) /* 0A000152 */
520         IL_0010: brtrue.s IL_0023
521
522         IL_0012: ldarg.0
523         IL_0013: ldfld native int testprint.KeyboardHook::HHookID /* 04000055 */
524         IL_0018: call int32 [mscorlib]System.IntPtr::op_Explicit(native int) /* 0A00010B */
525         IL_001d: call bool testprint.KeyboardHook::UnhookWindowsHookEx(int32) /* 060000E0 */
526         IL_0022: pop
527
528         IL_0023: ldarg.0
529         IL_0024: call instance void [mscorlib]System.Object::Finalize() /* 0A000159 */
530         IL_0029: ret
531     } // end of method KeyboardHook::Finalize
532
533     // Events
534     .event testprint.KeyboardHook/KeyDownEventHandler KeyDown
535     {
536         .addon void testprint.KeyboardHook::add_KeyDown(class
            testprint.KeyboardHook/KeyDownEventHandler)
537         .removeon void testprint.KeyboardHook::remove_KeyDown(class
            testprint.KeyboardHook/KeyDownEventHandler)
538     }
```

family: can be accessed by any class descending from the current one.

ldfld: loads the instance field onto the stack.
ldsfld: loads the static field onto the stack.

Cleaning up

Event declaration. We should remember that all events must have a subscribing method (.addon) and a unsubscribing method (.removeon), at least. ☺

```
.event testprint.KeyboardHook/KeyUpEventHandler KeyUp
{
    .addon void testprint.KeyboardHook::add_KeyUp(class
    testprint.KeyboardHook/KeyUpEventHandler)
    .removeon void testprint.KeyboardHook::remove_KeyUp(class
    testprint.KeyboardHook/KeyUpEventHandler)
}

.method /* 060000E3 */ public specialname static
    void add_KeyUp (
        class testprint.KeyboardHook/KeyUpEventHandler obj
    ) cil managed
{
    .custom instance void
    [mscorlib]System.Runtime.CompilerServices.CompilerGeneratedAttribute::.ctor(
    ) = (
        01 00 00 00
    )
    // Method begins at RVA 0xaabc
    // Code size 39 (0x27)
    .maxstack 3
    .locals /* 11000024 */ init (
        [0] class testprint.KeyboardHook/KeyUpEventHandler,
        [1] class testprint.KeyboardHook/KeyUpEventHandler,
        [2] class testprint.KeyboardHook/KeyUpEventHandler
    )
```

Declaring three local class variables in three different slots: 0, 1 and 2. We should remember that, eventually, slots of same type can be reused. However, it is another talk... ☺

```
24      IL_0000: ldsfld class testprint.KeyboardHook/KeyUpEventHandler
        testprint.KeyboardHook::KeyUpEvent /* 0400004D */
25      IL_0005: stloc.0
26      // loop start (head: IL_0006)
27          IL_0006: ldloc.0
28          IL_0007: stloc.1
29          IL_0008: ldloc.1
30          IL_0009: ldarg.0
31          IL_000a: call class [mscorlib]System.Delegate
            [mscorlib]System.Delegate::Combine(class [mscorlib]System.Delegate,
            class [mscorlib]System.Delegate) /* 0A000040 */
32          IL_000f: castclass testprint.KeyboardHook/KeyUpEventHandler /*
            0200001F */
33          IL_0014: stloc.2
34          IL_0015: ldsflda class testprint.KeyboardHook/KeyUpEventHandler
            testprint.KeyboardHook::KeyUpEvent /* 0400004D */
35          IL_001a: ldloc.2
36          IL_001b: ldloc.1
37          IL_001c: call !!0
            [mscorlib]System.Threading.Interlocked::CompareExchange<class
            testprint.KeyboardHook/KeyUpEventHandler>(!!0&, !!0, !!0) /* 2B000004 */
38          IL_0021: stloc.0
39          IL_0022: ldloc.0
40          IL_0023: ldloc.1
41          IL_0024: bne.un.s IL_0006
42      // end loop
43      IL_0026: ret
44  } // end of method KeyboardHook::add_KeyUp
```

Delegates are references representing "type-safe" function pointers. Thus, Combine( ) adds callback function pointers to an aggregate, which is used to implement the event.

Generic Delegate! Compares the second and third arguments and, if it's equal, replace the first argument (!!0&).

Finally, when the publisher calls the Invoke method of the aggregate delegate, so the event is raised. ☺

```
1  .method /* 06000001 */ assembly static
2      void Main (
3          string[] Args
4      ) cil managed noinlining nooptimization
5  {
6      .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = (
7          01 00 00 00
8      )
9      .custom instance void [mscorlib]System.Diagnostics.DebuggerHiddenAttribute::.ctor() = (
10         01 00 00 00
11     )
12     .custom instance void
       [System]System.ComponentModel.EditorBrowsableAttribute::.ctor(valuetype
       [System]System.ComponentModel.EditorBrowsableState) = (
13         01 00 02 00 00 00 00 00
14     )
15     // Method begins at RVA 0x2050
16     // Code size 22 (0x16)
17     .maxstack 8
18     .entrypoint
19
20     IL_0000: call bool
       [Microsoft.VisualBasic]Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase:
       :get_UseCompatibleTextRendering() /* 0A00001C */
21     IL_0005: call void
       [System.Windows.Forms]System.Windows.Forms.Application::SetCompatibleTextRenderingDefault(boo
       l) /* 0A00001D */
22     IL_000a: call class testprint.My.MyApplication testprint.My.MyProject::get_Application() /*
       06000007 */
23     IL_000f: ldarg.0
24     IL_0010: callvirt instance void
       [Microsoft.VisualBasic]Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase:
       :Run(string[]) /* 0A00001E */
25     IL_0015: ret
26 } // end of method MyApplication::Main
```

turn off compile optimization and not allow put this function as inline.

calling several instance contructors

calling the virtual method

```
1  .method /* 0600004C */ private
2      instance string Decrypt (
3          string sText
4      ) cil managed
5  {
6      // Method begins at RVA 0x2acc
7      // Code size 59 (0x3b)
8      .maxstack 4
9      .locals /* 11000005 */ init (
10         [0] int64,
11         [1] string,
12         [2] int64
13     )
14
15     IL_0000: ldnull
16     IL_0001: stloc.1
17     IL_0002: ldarg.1
18     IL_0003: call int32 [Microsoft.VisualBasic]Microsoft.VisualBasic.Strings::Len(string) /*
       0A000059 */
19     IL_0008: conv.i8
20     IL_0009: stloc.2
21     IL_000a: ldc.i4.1
22     IL_000b: conv.i8
23     IL_000c: stloc.0
24     IL_000d: br.s IL_0035
25     // loop start (head: IL_0035)
26         IL_000f: ldloc.1
27         IL_0010: ldarg.1
28         IL_0011: ldloc.0
29         IL_0012: conv.ovf.i4
30         IL_0013: ldc.i4.1
31         IL_0014: call string
           [Microsoft.VisualBasic]Microsoft.VisualBasic.Strings::Mid(string, int32, int32) /*
           0A00005A */
32         IL_0019: call int32 [Microsoft.VisualBasic]Microsoft.VisualBasic.Strings::Asc(string)
           /* 0A00005B */
33         IL_001e: ldc.i4.1
```

Function used to decrypt strings

declares and initializes the local variable

pushes a null references onto the stack

converts to int64 and pushes it onto the stack

loads the local variable onto stack.

converts to int32 and throws an exception when overflow

- ✓ DLL loaded from the Global Assembly Cache can and need to be monitored to detect strange behavior. Tools to log the DLL loading such as Fuslogvw.exe (Assembly Bind Log Viewer) and common applications such as Process Monitor can help us.

- ✓ Of course, .NET malware threats can try to compromise the .NET runtime class libraries and JIT, which would cause a deep infection in the system and demand a detailed investigation because:

  - ✓ changing the runtime library (at IL code) can be lethal to many applications.

  - ✓ it is feasible to change (hooking)/replace a runtime library.

  - ✓ Changing JIT cause same problems, but it is harder.

- ✓ Remember about basics:
  - ✓ copy DLL from GAC → dnSpy/Reflector + Reflexil → ildasm → change → ilasm → Ngen → copy back to GAC (malware dropper can accomplish this task) ☺

- ✓ Of course, nothing is so simple:
  - ✓ If the malware's target is a DLL from .NET runtime, so it is digitally signed and it would be necessary to have the private key to sign it. Unfortunately, we don't have.
  - ✓ Another option would be to generate a new pair of keys and re-sign all the DLL Framework. Unfortunately, it is so much work.
  - ✓ Copying a modified runtime DLL over the existing one can be difficult or almost impossible because other programs can be using it. Thus, we should stop programs and services to accomplish this task.
  - ✓ Eventually, it is necessary to reboot the machine (urgh!) to perform this copy from a script.
  - ✓ Using the new and modified DLL can be tricky: uninstall the existing native library (ngen uninstall <dll>) and remove it from its respective directory under NativeImages_<version> directory.
- ✓ There are other many tricks such as dropping an assembly into C:\Windows\System32 or Syswow64)\Tasks\Tasks.dll (hint from Casey Smith)

- ✓ An alternative would be change the Registry. In this case, the GAC continue being associated to the original (and untouched) assembly, while its associated native image is changed to the modified version.

- ✓ In this case:

  - ✓ HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Fusion\NativeImagesIndex\v2.0.50727_64\IL key holds information (name + signature key) about the original assembly.

  - ✓ HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Fusion\NativeImagesIndex\v2.0.50727_64\NI key would hold information (name + MVID) about the modified native image.

  - ✓ Using the MVID from NI key makes simple to locate the native image.

  - ✓ Thus, we can either override the native image with a modified version or change the MVID entry to point to another native image.

- ✓ GAC (old .NET assemblies) / GAC_32 (IL and x86) / GAC_64 (IL and x64) / GAC_MSIL (IL code) directories are under C:\Windows\Assembly directory.

- ✓ Most of the time, .NET malwares attacking the .NET libraries try either to remove some check or introduce hooking points at entry or exit of a method. In this case, System.Reflection is commonly used.

- ✓ Additionally, there are cases of .NET malware threats attacking applications and the service management offered by System.ServiceProcess.ServiceBase class and their associated method such as OnStart( ), OnStop( ), Run( ), ServiceMain( ) and so on.

- ✓ Modifying a target code for changing the execution flow demands inserting references (.assembly extern directive) to signed libraries (version + public key) to be able to access member and call methods.

- ✓ Of course, we should remember to increase the stack (.maxstack).

- ✓ At end, we have multiple types of attacks from a malicious managed code by establishing a C2, intercepting communication with trusted websites, opening shells, gathering system information, launching native second stage droppers, intercepting filesystem communication, stealing information and so on.

- ✓ WinDbg is always an excellent tool to understand a .NET malware in a better way or even getting a basic understanding, at least. ☺
- ✓ Install SOSEX extension:
  - ✓ Download it from http://www.stevestechspot.com/downloads/sosex_64.zip or http://www.stevestechspot.com/downloads/sosex_32.zip
  - ✓ Unpack it and copy to your WinDbg installation directory. For example: C:\Program Files (x86)\Windows Kits\10\Debuggers\x64|x86
- ✓ Attach the WinDbg to either a running application (the .NET malware) or a saved dump.
- ✓ Remember that the CLR process is composed by:
  - ✓ System Domain
  - ✓ Shared Domain
  - ✓ Default Domain
    - ✓ code running at this domains can't access resources from another application domain.

```
0:004> .loadby SOS.dll mscorwks
0:004> ln @$exentry
(712b7cef)   MSCOREE!ShellShim__CorExeMain   |  (712b7dd7)   MSCOREE!ShellShim_CreateConfigStream
Exact matches:
    MSCOREE!ShellShim__CorExeMain = <no type information>
0:004> u @$exentry
MSCOREE!ShellShim__CorExeMain:
712b7cef 8bff            mov     edi,edi
712b7cf1 55              push    ebp
712b7cf2 8bec            mov     ebp,esp
712b7cf4 51              push    ecx
712b7cf5 8365fc00        and     dword ptr [ebp-4],0
712b7cf9 8d45fc          lea     eax,[ebp-4]
712b7cfc 50              push    eax
712b7cfd e83595ffff      call    MSCOREE!GetShimImpl (712b1237)
0:004> u 011f0000+0x25B05A
MicroTik_Realtek_Driver+0x25b05a:
0144b05a ff2500201f01    jmp     dword ptr [MicroTik_Realtek_Driver+0x2000 (011f2000)]
0144b060 06              push    es
0144b061 07              pop     es
0144b062 06              push    es
0144b063 0504030201      add     eax,1020304h
0144b068 0102            add     dword ptr [edx],eax
0144b06a 03040506070809  add     eax,dword ptr [eax+9080706h]
0144b071 0a0b            or      cl,byte ptr [ebx]
0:004> dd 011f2000 L4
011f2000  712b4ddb 00000000 00000048 00050002
0:004> u 712b4ddb
MSCOREE!_CorExeMain_Exported:
712b4ddb 8bff            mov     edi,edi
712b4ddd 56              push    esi
712b4dde e80c2f0000      call    MSCOREE!ShellShim__CorExeMain (712b7cef)
712b4de3 6a00            push    0
712b4de5 8bf0            mov     esi,eax
712b4de7 e84bc4ffff      call    MSCOREE!GetShimImpl (712b1237)
712b4dec e93a800000      jmp     MSCOREE!_CorExeMain_Exported+0x11 (712bce2b)
MSCOREE!_imp_load__RegOpenKeyExW:
712b4df1 b82c102f71      mov     eax,offset MSCOREE!_imp__RegOpenKeyExW (712f102c)
```

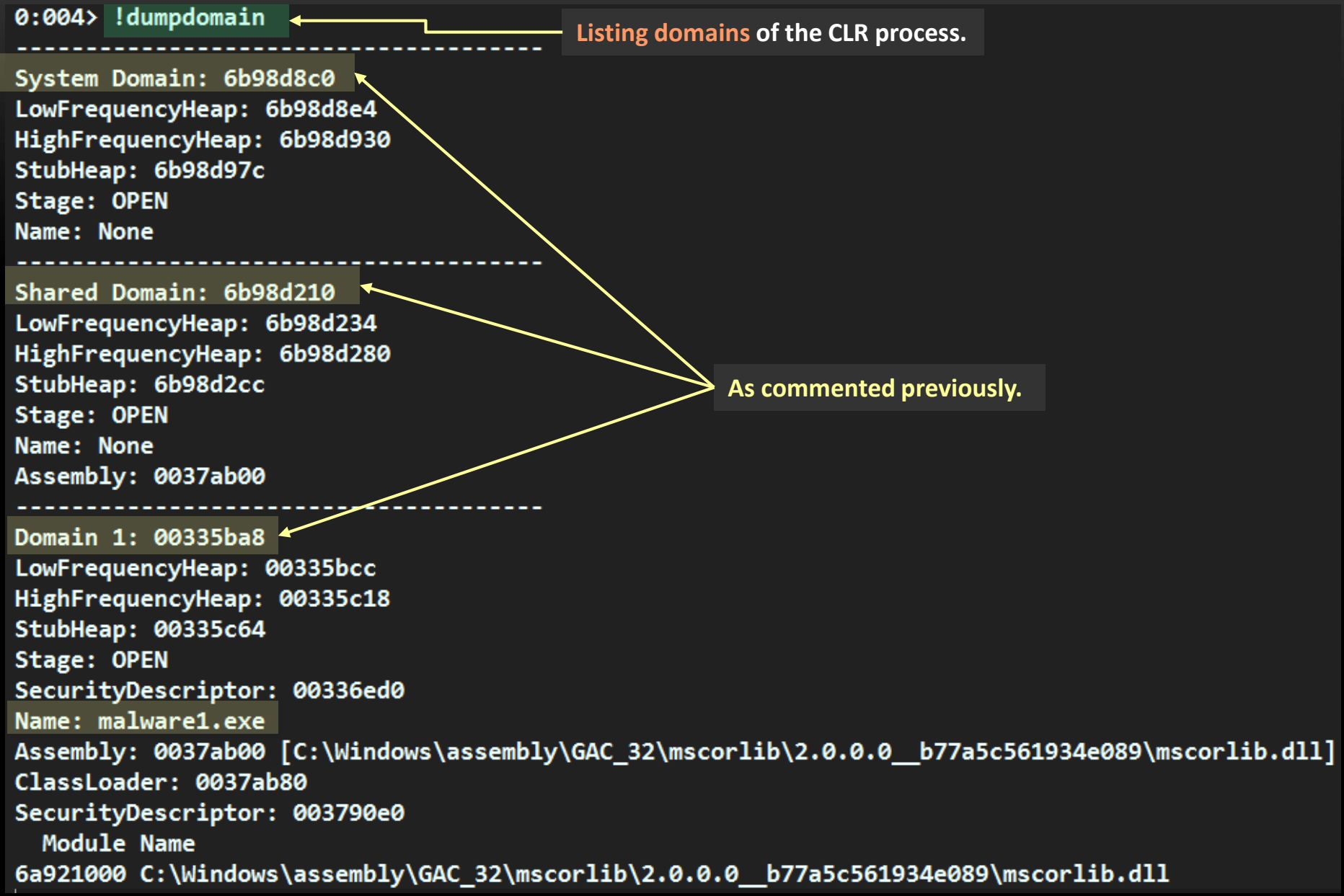
We could have used before this point: sxe ld mscorwks.dll ; g

❑ Remember:

- ✓ Malware executes
- ✓ Win loaders find the PE's entry point
- ✓ Jump to mscoree.dll
- ✓ Call to CorExeMain( )
- ✓ Return to assembly's entry point.

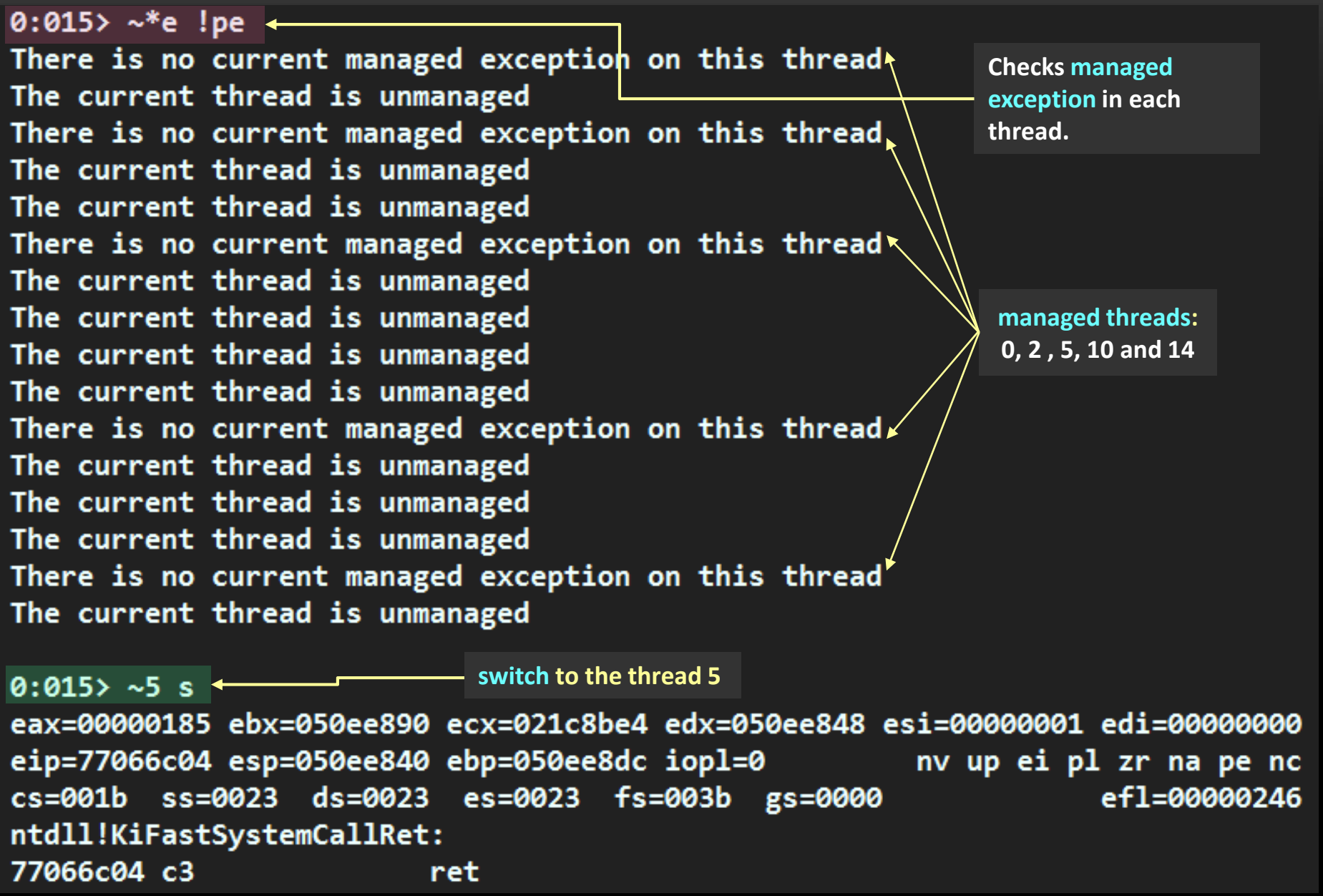
0x25B05A: Entry Point from dumpbin /headers malware1.exe

Disassembling CorExeMain( ) from the start.


0:004> !dumpdomain
Listing domains of the CLR process.
System Domain: 6b98d8c0
LowFrequencyHeap: 6b98d8e4
HighFrequencyHeap: 6b98d930
StubHeap: 6b98d97c
Stage: OPEN
Name: None
Shared Domain: 6b98d210
LowFrequencyHeap: 6b98d234
HighFrequencyHeap: 6b98d280
StubHeap: 6b98d2cc
Stage: OPEN
Name: None
Assembly: 0037ab00
As commented previously.
Domain 1: 00335ba8
LowFrequencyHeap: 00335bcc
HighFrequencyHeap: 00335c18
StubHeap: 00335c64
Stage: OPEN
SecurityDescriptor: 00336ed0
Name: malware1.exe
Assembly: 0037ab00 [C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll]
ClassLoader: 0037ab80
SecurityDescriptor: 003790e0
Module Name
6a921000 C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll

```
Assembly: 00384920 [C:\malware1.exe]
ClassLoader: 003849a0
SecurityDescriptor: 00384e80
  Module Name
00552c6c C:\malware1.exe

Assembly: 003868f8
[C:\Windows\assembly\GAC_MSIL\Microsoft.VisualBasic\8.0.0.0__b03f5f7f11d50a3a\Microsoft.VisualBasic.dll]
ClassLoader: 0038f710
SecurityDescriptor: 00325fc0

Assembly: 0038fd38 [C:\Windows\assembly\GAC_MSIL\System\2.0.0.0__b77a5c561934e089\System.dll]
ClassLoader: 0038fdf8
SecurityDescriptor: 0038f898

Assembly: 00390e30
[C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll]
ClassLoader: 003926d0
SecurityDescriptor: 00392468

Assembly: 00392db0 [C:\Windows\assembly\GAC_MSIL\System.Drawing\2.0.0.0__b03f5f7f11d50a3a\System.Drawing.dll]
ClassLoader: 00392e70
SecurityDescriptor: 003928d8

Assembly: 00394708
[C:\Windows\assembly\GAC_MSIL\System.Runtime.Remoting\2.0.0.0__b77a5c561934e089\System.Runtime.Remoting.dll]
ClassLoader: 00394788
SecurityDescriptor: 00394960

Assembly: 0039fb48
[C:\Windows\assembly\GAC_MSIL\System.Management\2.0.0.0__b03f5f7f11d50a3a\System.Management.dll]
ClassLoader: 0039fbc8
SecurityDescriptor: 0039ff98
```

Used assemblies tell us a bit about the application.


0:015> ~*e !pe
There is no current managed exception on this thread
The current thread is unmanaged
There is no current managed exception on this thread
The current thread is unmanaged
The current thread is unmanaged
There is no current managed exception on this thread
The current thread is unmanaged
The current thread is unmanaged
The current thread is unmanaged
The current thread is unmanaged
There is no current managed exception on this thread
The current thread is unmanaged
The current thread is unmanaged
The current thread is unmanaged
There is no current managed exception on this thread
The current thread is unmanaged
Checks managed exception in each thread.
managed threads: 0, 2 , 5, 10 and 14
switch to the thread 5
0:015> ~5 s
eax=00000185 ebx=050ee890 ecx=021c8be4 edx=050ee848 esi=00000001 edi=00000000
eip=77066c04 esp=050ee840 ebp=050ee8dc iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
ntdll!KiFastSystemCallRet:
77066c04 c3 ret

```
0:005> ~0 s
eax=00001273 ebx=020529b8 ecx=020529b8 edx=001df1bc esi=020ef6ac edi=02252818
eip=77066c04 esp=001df1b4 ebp=001df248 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!KiFastSystemCallRet:
77066c04 c3              ret
0:000> !ClrStack
OS Thread Id: 0xf18 (0)
ESP       EIP
001df1c0 77066c04 [InlinedCallFrame: 001df1c0]
System.Windows.Forms.UnsafeNativeMethods.WaitMessage()
001df1bc 669b8e08
System.Windows.Forms.Application+ComponentManager.System.Windows.Forms.UnsafeNativeMethods
.IMsoComponentManager.FPushMessageLoop(Int32, Int32, Int32)
001df258 669b88f7
System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner(Int32,
System.Windows.Forms.ApplicationContext)
001df2ac 669b8741 System.Windows.Forms.Application+ThreadContext.RunMessageLoop(Int32,
System.Windows.Forms.ApplicationContext)
001df2dc 66ece818
System.Windows.Forms.Application.Run(System.Windows.Forms.ApplicationContext)
001df2e8 68fd28f1
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.OnRun()
001df314 68fd2f5f
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.DoApplicationModel()
001df340 68fd19c8
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Run(System.String[])
001df39c 00690095 testprint.My.MyApplication.Main(System.String[])
001df5c0 6b421b8c [GCFrame: 001df5c0]
```

switch to the thread 0

Check the managed stack trace for this thread.

```
0:000> !Threads
ThreadCount: 5
UnstartedThread: 0
BackgroundThread: 4
PendingThread: 0
DeadThread: 0
Hosted Runtime: no
                                    PreEmptive   GC Alloc                 Lock
       ID OSID ThreadOBJ    State     GC       Context       Domain     Count APT Exception
   0    1  f18 0036b8a0     6020 Enabled  0225283c:0225357c 00335ba8     0 STA
   2    2  ed4 00376998     b220 Enabled  021a95a8:021ab57c 00335ba8     0 MTA (Finalizer)
   5    3  e9c 003b1418   200b220 Enabled  00000000:00000000 00335ba8     0 MTA
  10    5  3a8 003da178      220 Enabled  00000000:00000000 00335ba8     0 Ukn
  14    7  d20 003f4fe8   200b220 Enabled  00000000:00000000 00335ba8     1 MTA
0:000> kn
 # ChildEBP RetAddr
00 001df1b0 76dc6699 ntdll!KiFastSystemCallRet
01 001df1b4 669b8e08 USER32!NtUserWaitMessage+0xc
02 001df248 669b88f7 System_Windows_Forms_ni+0x208e08
03 001df2a0 669b8741 System_Windows_Forms_ni+0x2088f7
04 001df2d0 66ece818 System_Windows_Forms_ni+0x208741
05 001df2e0 68fd28f1 System_Windows_Forms_ni+0x71e818
06 001df30c 68fd2f5f Microsoft_VisualBasic_ni+0x1128f1
07 001df338 68fd19c8 Microsoft_VisualBasic_ni+0x112f5f
08 001df3a0 6b421b8c Microsoft_VisualBasic_ni+0x1119c8
09 001df3b0 6b4385ab mscorwks!CallDescrWorker+0x33
0a 001df430 6b44064b mscorwks!CallDescrWorkerWithHandler+0xa3
0b 001df56c 6b44067e mscorwks!MethodDesc::CallDescr+0x19c
0c 001df588 6b44069c mscorwks!MethodDesc::CallTargetWorker+0x1f
0d 001df5a0 6b552fe5 mscorwks!MethodDescCallSite::Call_RetArgSlot+0x1a
0e 001df704 6b552f05 mscorwks!ClassLoader::RunMain+0x223
0f 001df96c 6b553422 mscorwks!Assembly::ExecuteMainMethod+0xa6
10 001dfe3c 6b55360c mscorwks!SystemDomain::ExecuteMainMethod+0x45e
11 001dfe8c 6b55353c mscorwks!ExecuteEXE+0x59
```

Get a list of managed threads. Of course, we could used the -special option to get additional information.

COM Threading Model:

- ✓ STA: Single Thread Apartment
- ✓ MTA: Multi Thread Apartment

Checks the unmanaged stack trace for this thread.

Threat state:

- ✓ (0x0) Newly initialized thread.
- ✓ (0x020) It can enter a Join.
- ✓ (0x200) background thread.

```
0:000> !IP2MD Microsoft_VisualBasic_ni+0x1119c8
MethodDesc: 68ed4f80
Method Name: Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Run(System.String[])
Class: 68ecbd68
MethodTable: 68ffe670
mdToken: 0600005e
Module: 68ec1000
IsJitted: yes
CodeAddr: 68fd1960
0:000> !U 68ed4f80
preJIT generated code
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Run(System.String[])
Begin 68fd1960, size 48f. Cold region begin 69019908, size 4f
Hot region:
68fd1960 55              push    ebp
68fd1961 8bec            mov     ebp,esp
68fd1963 57              push    edi
68fd1964 56              push    esi
68fd1965 53              push    ebx
68fd1966 83ec48          sub     esp,48h
68fd1969 8bf1            mov     esi,ecx
68fd196b 8d7dd4          lea     edi,[ebp-2Ch]
68fd196e b908000000      mov     ecx,8
68fd1973 33c0            xor     eax,eax
68fd1975 f3ab            rep stos dword ptr es:[edi]
68fd1977 8bce            mov     ecx,esi
68fd1979 33c0            xor     eax,eax
68fd197b 8945e4          mov     dword ptr [ebp-1Ch],eax
68fd197e 894dd0          mov     dword ptr [ebp-30h],ecx
68fd1981 8bfa            mov     edi,edx
68fd1983 8b75d0          mov     esi,dword ptr [ebp-30h]
68fd1986 8d55f0          lea     edx,[ebp-10h]
68fd1989 b9804fed68      mov     ecx,offset Microsoft_VisualBasic_ni+0x14f80 (68ed4f80) (MD:
```

Check if the instruction pointer address belongs to the JIT code and find the Method Descriptor ☺

Disassembling the code

```
0:000> !ClrStack
OS Thread Id: 0xf18 (0)
ESP       EIP
001df1c0 77066c04 [InlinedCallFrame: 001df1c0] System.Windows.Forms.UnsafeNativeMethods.WaitMessage()
001df1bc 669b8e08
System.Windows.Forms.Application+ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IMsoComponentManager
.FPushMessageLoop(Int32, Int32, Int32)
001df258 669b88f7 System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner(Int32,
System.Windows.Forms.ApplicationContext)
001df2ac 669b8741 System.Windows.Forms.Application+ThreadContext.RunMessageLoop(Int32,
System.Windows.Forms.ApplicationContext)
001df2dc 66ece818 System.Windows.Forms.Application.Run(System.Windows.Forms.ApplicationContext)
001df2e8 68fd28f1 Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.OnRun()
001df314 68fd2f5f Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.DoApplicationModel()
001df340 68fd19c8 Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Run(System.String[])
001df39c 00690095 testprint.My.MyApplication.Main(System.String[])
001df5c0 6b421b8c [GCFrame: 001df5c0]
0:000> !IP2MD 669b88f7
MethodDesc: 6686bc44
Method Name: System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner(Int32,
System.Windows.Forms.ApplicationContext)
Class: 667e6af8
MethodTable: 66a01648
mdToken: 06001329
Module: 667b1000
IsJitted: yes
CodeAddr: 669b8780
0:000> !DumpMD 6686bc44
Method Name: System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner(Int32,
System.Windows.Forms.ApplicationContext)
Class: 667e6af8
MethodTable: 66a01648
mdToken: 06001329
Module: 667b1000
IsJitted: yes
CodeAddr: 669b8780
```

Checks the managed stack

Check if the instruction pointer address belongs to the JIT code.

Method definition.

Remember: Metadata token is composed by a Table Reference (1 byte) and a Table Index (3 byte).

Display information about the MethodDesc structure

```
0:000> !DumpClass 667e6af8
```

Displays information about the EEClass structure associated with a type

```
Class Name: System.Windows.Forms.Application+ThreadContext
mdToken: 020001e4
(C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll)
Parent Class: 6a924160
Module: 667b1000
Method Table: 66a01648
Vtable Slots: 12
Total Method Slots: 3c
Class Attributes: 100105
NumInstanceFields: 1b
NumStaticFields: 5
ThreadStaticOffset: 3
ThreadStaticsSize: 4
      MT    Field   Offset                 Type VT     Attr    Value Name
6ab92698  400018a        4       System.Object  0 instance          __identity
67528ab8  4001054        8 ...ptionEventHandler  0 instance          threadExceptionHandler
6ab8bce8  4001055        c  System.EventHandler  0 instance          idleHandler
6ab8bce8  4001056       10  System.EventHandler  0 instance          enterModalHandler
6ab8bce8  4001057       14  System.EventHandler  0 instance          leaveModalHandler
66a05f94  4001058       18 ...pplicationContext  0 instance          applicationContext
66a10890  4001059       1c ...ion+ParkingWindow  0 instance          parkingWindow
669ff730  400105a       20 ...ows.Forms.Control  0 instance          marshalingControl
6ab95678  400105b       24 ...ation.CultureInfo  0 instance          culture
6ab94aac  400105c       28 ...ections.ArrayList  0 instance          messageFilters
6ab6624c  400105d       2c      System.Object[]  0 instance          messageFilterSnapshot
6ab95344  400105e       44        System.IntPtr  1 instance          handle
6ab94cc8  400105f       48         System.Int32  1 instance          id
6ab94cc8  4001060       4c         System.Int32  1 instance          messageLoopCount
6ab94cc8  4001061       50         System.Int32  1 instance          threadState
6ab94cc8  4001062       54         System.Int32  1 instance          modalCount
6ab8c22c  4001063       30 System.WeakReference  0 instance          activatingControlRef
66a0548c  4001064       34 ...oComponentManager  0 instance          componentManager
6ab66548  4001065       60       System.Boolean  1 instance          externalComponentManager
6ab66548  4001066       61       System.Boolean  1 instance          fetchingComponentManager
6ab94cc8  4001067       58         System.Int32  1 instance          componentID
669fd874  4001068       38 ...indows.Forms.Form  0 instance          currentForm
```

```
669ea16c  4001069        3c ...ion+ThreadWindows  0 instance           threadWindows
66a059bc  400106a        64 ...NativeMethods+MSG  1 instance           tempMsg
6ab94cc8  400106b        5c          System.Int32  1 instance           disposeCount
6ab66548  400106c        62       System.Boolean  1 instance           ourModalLoop
6714b010  400106d        40 ...ssageLoopCallback  0 instance           messageLoopCallback
6ab950d0  400104f      17c ...ections.Hashtable  0   static 020528e4 contextHash
6ab92698  4001050      180        System.Object  0   static 020529ac tcInternalSyncObject
6ab94cc8  4001051      998          System.Int32  1   static        1 totalMessageLoopCount
6ab94cc8  4001052      99c          System.Int32  1   static       -1 baseLoopReason
66a01648  4001053        0 ...ion+ThreadContext  0 TLstatic  currentThreadContext
    >> Thread:Value.f18:020529b8 <<
0:000> !dumpvc 6ab950d0 020528e4
Name: System.Collections.Hashtable
MethodTable 6ab950d0
EEClass: 6a94e9cc
Size: 56(0x38) bytes
 (C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll)
Fields:
      MT    Field   Offset                 Type VT     Attr    Value Name
6ab951cc  400093e        0 ...ashtable+bucket[]  0 instance 6ab950d0 buckets
6ab94cc8  400093f       18          System.Int32  1 instance        0 count
6ab94cc8  4000940       1c          System.Int32  1 instance        1 occupancy
6ab94cc8  4000941       20          System.Int32  1 instance        0 loadsize
6ab8c2c0  4000942       24         System.Single  1 instance 0.000000 loadFactor
6ab94cc8  4000943       28          System.Int32  1 instance 1060655596 version
6ab66548  4000944       2c       System.Boolean  1 instance        1 isWriterInProgress
6ab94f4c  4000945        4 ...tions.ICollection  0 instance 0205291c keys
6ab94f4c  4000946        8 ...tions.ICollection  0 instance 00000000 values
6ab8e854  4000947        c ...IEqualityComparer  0 instance 00000000 _keycomparer
6ab92698  4000948       10        System.Object  0 instance 00000000 _syncRoot
6ab80e74  4000949       14 ...SerializationInfo  0 instance 00000000 m_siInfo
```

dumps the object content, but in this case it is a value type. ☺

Class

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

```
0:000> !DumpMT 66a01648
EEClass: 667e6af8
Module: 667b1000
Name: System.Windows.Forms.Application+ThreadContext
mdToken: 020001e4
(C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll)
BaseSize: 0x84
ComponentSize: 0x0
Number of IFaces in IFaceMap: 1
Slots in VTable: 65
0:000> !DumpMT -md 66a01648
EEClass: 667e6af8
Module: 667b1000
Name: System.Windows.Forms.Application+ThreadContext
mdToken: 020001e4
(C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll)
BaseSize: 0x84
ComponentSize: 0x0
Number of IFaces in IFaceMap: 1
Slots in VTable: 65
--------------------------------------
MethodDesc Table
   Entry MethodDesc    JIT Name
6aae7ae0   6a9649d8    PreJIT System.Object.ToString()
6aae7b00   6a9649e0    PreJIT System.Object.Equals(System.Object)
6aae7b70   6a964a10    PreJIT System.Object.GetHashCode()
6697d260   6686bb74    PreJIT System.Windows.Forms.Application+ThreadContext.Finalize()
6aff2d30   6a96d5b4    PreJIT System.MarshalByRefObject.GetLifetimeService()
66ecf890   6686bbf4    PreJIT System.Windows.Forms.Application+ThreadContext.InitializeLifetimeService()
6aafbe90   6a96d5c4    PreJIT System.MarshalByRefObject.CreateObjRef(System.Type)
66ecfda0   6686bc98    PreJIT
System.Windows.Forms.Application+ThreadContext.System.Windows.Forms.UnsafeNativeMethods.IMsoComponent.FDebugM
essage(IntPtr, Int32, IntPtr, IntPtr)
669b90c0   6686bca0    PreJIT
System.Windows.Forms.Application+ThreadContext.System.Windows.Forms.UnsafeNativeMethods.IMsoComponent.FPreTra
nslateMessage(MSG ByRef)
66ecfdc0   6686bca8    PreJIT
```

Dump information about the Method Table

Type definition

Dump information about the Method Table and display a list of all methods.

EEClass data structure is similar to the method table, but it stores fields that are less frequently used.

Code is PreJIT compiled

```
0:000> !DumpModule 667b1000
Name: C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll
Attributes: PEFile
Assembly: 00390e30
LoaderHeap: 00000000
TypeDefToMethodTableMap: 668bf8c4
TypeRefToMethodTableMap: 668bc8e8
MethodDefToDescMap: 668c1c0c
FieldDefToDescMap: 668dd114
MemberRefToDescMap: 668bd380
FileReferencesMap: 667b2f14
AssemblyReferencesMap: 667b2f18
MetaData start address: 66ab9aa8 (2575412 bytes)
0:000> !DumpAssembly 00390e30
Parent Domain: 00335ba8
Name: C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll
ClassLoader: 003926d0
SecurityDescriptor: 004f4fe8
  Module Name
667b1000 C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll
0:000> !DumpDomain 00335ba8
-----------------------------------
Domain 1: 00335ba8
LowFrequencyHeap: 00335bcc
HighFrequencyHeap: 00335c18
StubHeap: 00335c64
Stage: OPEN
SecurityDescriptor: 00336ed0
Name: malware1.exe
Assembly: 0037ab00 [C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll]
ClassLoader: 0037ab80
SecurityDescriptor: 003790e0
  Module Name
6a921000 C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll
005823d8 C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\sortkey.nlp
00582088 C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\sorttbls.nlp
```

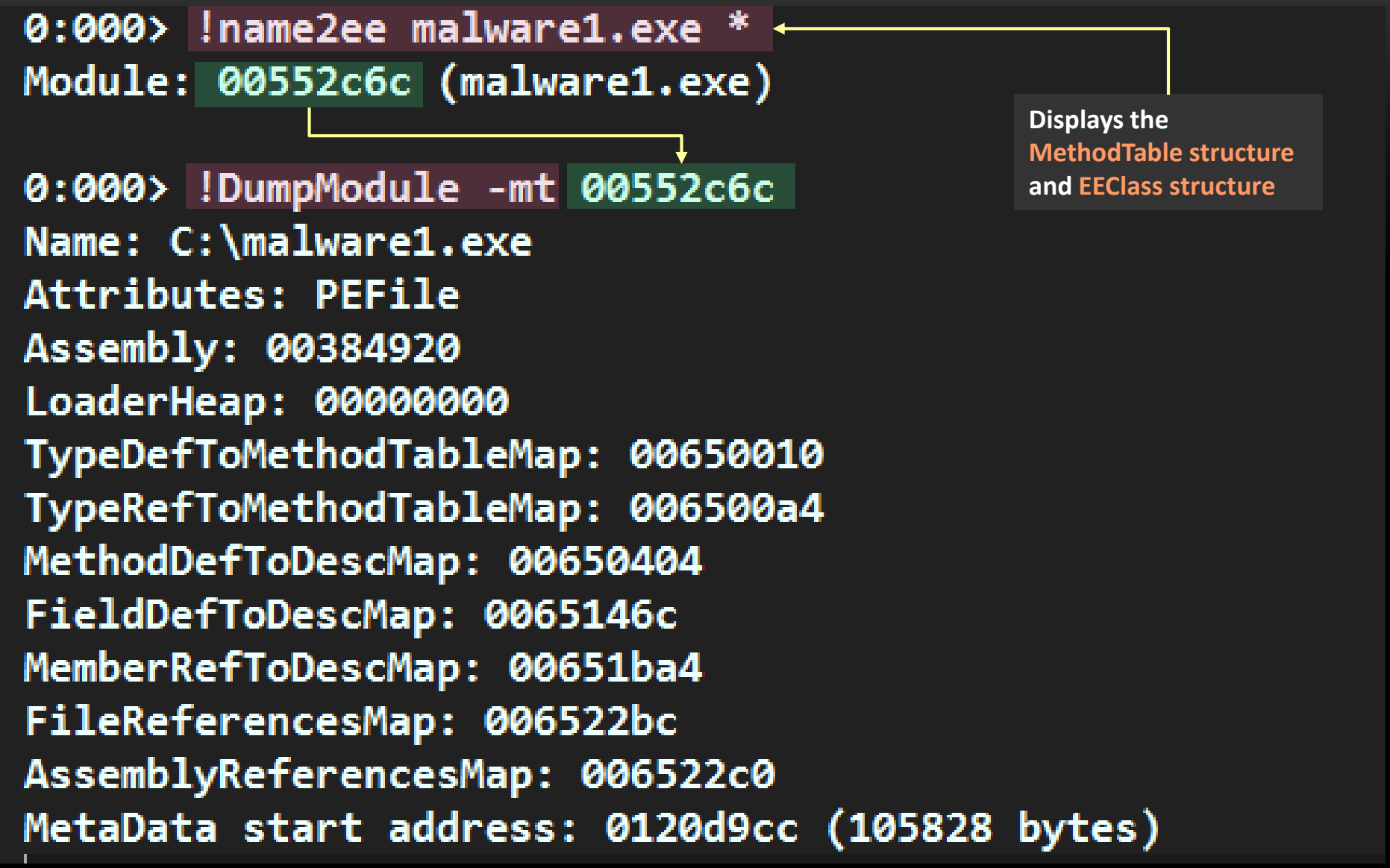
Dump information about a specific module

Dump information about the assembly (as shown previously)

Data accessed and/or updated less frequently

Data accessed and/or updated very frequently

Data used to help COM operations


0:000> !name2ee malware1.exe *
Module: 00552c6c (malware1.exe)
0:000> !DumpModule -mt 00552c6c
Name: C:\malware1.exe
Attributes: PEFile
Assembly: 00384920
LoaderHeap: 00000000
TypeDefToMethodTableMap: 00650010
TypeRefToMethodTableMap: 006500a4
MethodDefToDescMap: 00650404
FieldDefToDescMap: 0065146c
MemberRefToDescMap: 00651ba4
FileReferencesMap: 006522bc
AssemblyReferencesMap: 006522c0
MetaData start address: 0120d9cc (105828 bytes)
Displays the MethodTable structure and EEClass structure

```
Types defined in this module

      MT    TypeDef Name
-----------------------------------------------------------------------------
00553f3c 0x02000002 testprint.My.MyApplication
005560f4 0x02000003 testprint.My.MyComputer
00555de8 0x02000004 testprint.My.MyProject
005589c8 0x02000005 testprint.My.Resources.Resources
00557cf0 0x02000008 testprint.Client
00557bb4 0x02000009 testprint.cTripleDES
00557138 0x0200000c testprint.Form2
00558164 0x0200000d testprint.KeyboardHook
00558a50 0x0200000e testprint.libera
00558004 0x02000010 testprint.UploadScreenshot
00557ae0 0x02000011 <PrivateImplementationDetails>
00556494 0x02000012 testprint.My.MyProject+MyForms
005565a8 0x02000013 testprint.My.MyProject+MyWebServices
00556068 0x02000014 testprint.My.MyProject+ThreadSafeObjectProvider`1
0055820c 0x02000015 testprint.Client+NeuerTextIstDaEventHandler
00559dac 0x0200001c testprint.KeyboardHook+KBDLLHOOKSTRUCT
00559cc8 0x0200001d testprint.KeyboardHook+KBDLLHOOKSTRUCTFlags
00558410 0x0200001e testprint.KeyboardHook+KeyDownEventHandler
00559e50 0x0200001f testprint.KeyboardHook+KeyUpEventHandler
0055832c 0x02000020 testprint.KeyboardHook+KBDLLHookProc
00557aa0 0x02000024 <PrivateImplementationDetails>+__StaticArrayInitTypeSize=24
```

Bad intentions? ☺

```
0:000> !name2ee malware1.exe testprint.Client
Module: 00552c6c (malware1.exe)
Token: 0x02000008
MethodTable: 00557cf0
EEClass: 00654ee4
Name: testprint.Client
0:000> !dumpmt -md 00557cf0
EEClass: 00654ee4
Module: 00552c6c
Name: testprint.Client
mdToken: 02000008  (C:\malware1.exe)
BaseSize: 0x2c
ComponentSize: 0x0
Number of IFaces in IFaceMap: 0
Slots in VTable: 14
--------------------------------------
MethodDesc Table
   Entry MethodDesc    JIT Name
6aae7ae0   6a9649d8 PreJIT System.Object.ToString()
6aae7b00   6a9649e0 PreJIT System.Object.Equals(System.Object)
6aae7b70   6a964a10 PreJIT System.Object.GetHashCode()
6ab59510   6a964a34 PreJIT System.Object.Finalize()
0055c4a5   00557c70   NONE testprint.Client..ctor()
00690870   00557c78    JIT testprint.Client.add_NeuerTextIstDa(NeuerTextIstDaEventHandler)
0055c4ad   00557c84   NONE testprint.Client.remove_NeuerTextIstDa(NeuerTextIstDaEventHandler)
0055c4b1   00557c90   NONE testprint.Client.Verbinden(System.String, Int32)
0055c4b5   00557c9c   NONE testprint.Client.Disconnect()
0055c4b9   00557ca8   NONE testprint.Client.Reconnect()
0055c4bd   00557cb4   NONE testprint.Client.TextEmpfangen()
0055c4c1   00557cc0   NONE testprint.Client.Textsenden(System.String)
0055c4c5   00557ccc   NONE testprint.Client.Decrypt(System.String)
0055c4c9   00557cd8   NONE testprint.Client.Encrypt(System.String)
```

PreJIT: pre-compiled code
JIT: compiled Code
NONE: the code hasn't been compiled by the JIT.

```
0:000> !dumpclass 00654ee4
Class Name: testprint.Client
mdToken: 02000008 (C:\malware1.exe)
Parent Class: 6a923ef0
Module: 00552c6c
Method Table: 00557cf0
Vtable Slots: 4
Total Method Slots: 5
Class Attributes: 1
NumInstanceFields: 9
NumStaticFields: 0
      MT    Field   Offset                 Type VT     Attr    Value Name
679a7cc8  400000b        4 ...Sockets.TcpClient  0 instance          TcpClient
6ab7a9fc  400000c        8 ...m.IO.StreamReader  0 instance          Str
6ab96264  400000d        c ...m.IO.StreamWriter  0 instance          Stw
6ab92a7c  400000e       10        System.String  0 instance          IP
6ab94cc8  400000f       20         System.Int32  1 instance          Port
6ab66548  4000010       24       System.Boolean  1 instance          Verbunden
6ab92a7c  4000011       14        System.String  0 instance          casa
6ab92a7c  4000012       18        System.String  0 instance          Active_windows
0055820c  4000013       1c ...IstDaEventHandler  0 instance          NeuerTextIstDaEvent
0:000> !name2ee malware1.exe testprint.Client.Verbinden
Module: 00552c6c (malware1.exe)
Token: 0x06000047
MethodDesc: 00557c90
Name: testprint.Client.Verbinden(System.String, Int32)
Not JITTED yet. Use !bpmd -md 00557c90 to break on run.
0:000> !dumpmd 00557c90
Method Name: testprint.Client.Verbinden(System.String, Int32)
Class: 00654ee4
MethodTable: 00557cf0
mdToken: 06000047
Module: 00552c6c
IsJitted: no
CodeAddr: ffffffff
0:000> !bpmd -md 00557c90
MethodDesc = 00557c90
Adding pending breakpoints.
```

Shows information about the EEClass structure

Displays the MethodTable structure and EEClass structure of test.Client.Verbunden method.

Displays the MethodDesc structure information

Set up a breakpoint on a code that is not JIT yet. ☺

```
0:000> !DumpStackObjects
OS Thread Id: 0xf18 (0)
ESP/REG  Object   Name
ebx      020529b8 System.Windows.Forms.Application+ThreadContext
ecx      020529b8 System.Windows.Forms.Application+ThreadContext
esi      020ef6ac System.Windows.Forms.Application+ComponentManager+ComponentHashtableEntry
edi      02252818 System.Collections.Hashtable+HashtableEnumerator
001df1e8 0217a36c System.Windows.Forms.NativeMethods+MSG[]
001df1ec 020529b8 System.Windows.Forms.Application+ThreadContext
001df1f4 020ef664 System.Windows.Forms.Application+ComponentManager
001df26c 020529b8 System.Windows.Forms.Application+ThreadContext
001df2a8 02052888 Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase+WinFormsAppContext
001df2e8 02052714 testprint.My.MyApplication
001df304 02052714 testprint.My.MyApplication
001df314 02052714 testprint.My.MyApplication
001df334 02052604 System.Object
001df39c 02052604 System.Object
001df454 02052604 System.Object
001df604 02052604 System.Object
001df62c 02051e48 System.Object[]    (System.String[])
0:000> !DumpArray -details 0217a36c
Name: System.Windows.Forms.NativeMethods+MSG[]
MethodTable: 66a0590c
EEClass: 667e89b8
Size: 40(0x28) bytes
Array: Rank 1, Number of elements 1, Type VALUETYPE
Element Methodtable: 66a059bc
[0] 0217a374
    Name: System.Windows.Forms.NativeMethods+MSG
    MethodTable 66a059bc
    EEClass: 667bd880
    Size: 36(0x24) bytes
     (C:\Windows\assembly\GAC_MSIL\System.Windows.Forms\2.0.0.0__b77a5c561934e089\System.Windows.Forms.dll)
    Fields:
          MT    Field   Offset                 Type VT     Attr    Value Name
    6ab95344  4002ba2        0        System.IntPtr  1 instance    40210 hwnd
    6ab94cc8  4002ba3        4         System.Int32  1 instance      275 message
    6ab95344  4002ba4        8        System.IntPtr  1 instance        4 wParam
    6ab95344  4002ba5        c        System.IntPtr  1 instance        0 lParam
    6ab94cc8  4002ba6       10         System.Int32  1 instance 10047883 time
    6ab94cc8  4002ba7       14         System.Int32  1 instance      248 pt_x
```

Performs stack walking and display managed objects from current thread

Dumps out arrays

```
0:000> !DumpObj 02052714
Name: testprint.My.MyApplication
MethodTable: 00553f3c
EEClass: 00653758
Size: 108(0x6c) bytes
 (C:\malware1.exe)
Fields:
```

Method Table of the field

Value Type: 1
Reference: 0

MT	Field	Offset	Type	VT	Attr	Value	Name
68ffced8	4000001	4	...Basic.Logging.Log	0	instance	00000000	m_Log
68ffcf28	4000002	8	...ices.AssemblyInfo	0	instance	00000000	m_Info
00000000	400000a	c		0	instance	02055644	m_CommandLineArgs
68ff713c	400001f	10	...artupEventHandler	0	instance	00000000	StartupEvent
68ff71c8	4000020	14	...tanceEventHandler	0	instance	00000000	StartupNextInstanceEvent
68ff7254	4000021	18	...tdownEventHandler	0	instance	00000000	ShutdownEvent
6ab94aac	4000024	1c	...ections.ArrayList	0	instance	00000000	m_UnhandledExceptionHandlers
6ab66548	4000025	60	System.Boolean	1	instance	0	m_ProcessingUnhandledExceptionEvent
6ab66548	4000026	61	System.Boolean	1	instance	0	m_TurnOnNetworkListener
6ab66548	4000027	62	System.Boolean	1	instance	1	m_FinishedOnInitilaize
6ab94aac	4000028	20	...ections.ArrayList	0	instance	00000000	m_NetworkAvailabilityEventHandlers
6ab7993c	4000029	24	...g.EventWaitHandle	0	instance	00000000	m_FirstInstanceSemaphore
6ab7993c	400002a	28	...g.EventWaitHandle	0	instance	00000000	m_MessageRecievedSemaphore
68ffe6d8	400002b	2c	...c.Devices.Network	0	instance	00000000	m_NetworkObject
6ab92a7c	400002c	30	System.String	0	instance	00000000	m_MemoryMappedID
6ab90a48	400002d	34	...es.SafeFileHandle	0	instance	00000000	m_FirstInstanceMemoryMappedFileHandle
6ab66548	400002e	63	System.Boolean	1	instance	0	m_IsSingleInstance
68ff7088	400002f	58	System.Int32	1	instance	0	m_ShutdownStyle
6ab66548	4000030	64	System.Boolean	1	instance	1	m_EnableVisualStyles
6ab66548	4000031	65	System.Boolean	1	instance	1	m_DidSplashScreen
6ab66548	4000032	66	System.Boolean	1	instance	1	m_Ok2CloseSplashScreen
669fd874	4000033	38	...indows.Forms.Form	0	instance	00000000	m_SplashScreen
6ab94cc8	4000034	5c	System.Int32	1	instance	2000	m_MinimumSplashExposure
679ad95c	4000035	3c	System.Timers.Timer	0	instance	00000000	m_SplashTimer
6ab92698	4000036	40	System.Object	0	instance	020527b8	m_SplashLock
68ffe728	4000037	44	...inFormsAppContext	0	instance	02052888	m_AppContext
6ab8bd60	4000038	48	...ronizationContext	0	instance	020528b4	m_AppSyncronizationContext
6ab92698	4000039	4c	System.Object	0	instance	020527c4	m_NetworkAvailChangeLock
6ab66548	400003a	67	System.Boolean	1	instance	1	m_SaveMySettingsOnExit
6ab751d8	400003b	50	...endOrPostCallback	0	instance	00000000	m_StartNextInstanceCallback
6ab92a7c	400003c	54	System.String	0	instance	00000000	m_HostName

```
0:000> !eeheap -gc
Number of GC Heaps: 1
generation 0 starts at 0x0218f57c
generation 1 starts at 0x0205100c
generation 2 starts at 0x02051000
ephemeral segment allocation context: none
 segment    begin allocated     size
02050000 02051000  02253588 0x00202588(2106760)
Large object heap starts at 0x03051000
 segment    begin allocated     size
03050000 03051000  033392f8 0x002e82f8(3048184)
Total Size  0x4ea880(5154944)
------------------------------
GC Heap Size  0x4ea880(5154944)
```

As a general overview, during allocation requests:

- ✓ If the maximum expected memory for the Gen0 is exceeded, collect non-rooted objects and promote rooted objects to Gen 1.
- ✓ The same approach is valid when collecting objects from Gen 1 and Gen 2.
- ✓ If Gen 2 is exceeded, so GC adds a new segment to Gen 2.
- ✓ Objects in Gen 0 and 1 are short-lived.

```
0:000> !gcroot 0x02052714
```

from the previous slide ☺

```
Note: Roots found on stacks may be false positives. Run "!help gcroot" for
more info.
ebx:Root:020529b8(System.Windows.Forms.Application+ThreadContext)->
02052888(Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase+WinFormsAppContext)->
02052714(testprint.My.MyApplication)
Scan Thread 0 OSTHread f18
ESP:1df2a8:Root:02052888(Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase+WinFor
msAppContext)
ESP:1df2e8:Root:02052714(testprint.My.MyApplication)
ESP:1df304:Root:02052714(testprint.My.MyApplication)
ESP:1df314:Root:02052714(testprint.My.MyApplication)
Scan Thread 2 OSTHread ed4
Scan Thread 5 OSTHread e9c
Scan Thread 10 OSTHread 3a8
Scan Thread 14 OSTHread d20
DOMAIN(00335BA8):HANDLE(Strong):5311f8:Root:02051a04(System.Threading.Thread)->
02052610(System.Object[][])->
02052624(System.Object[])->
02052714(testprint.My.MyApplication)
```

from stack...

Reference chain to the object ☺

from handle tables...

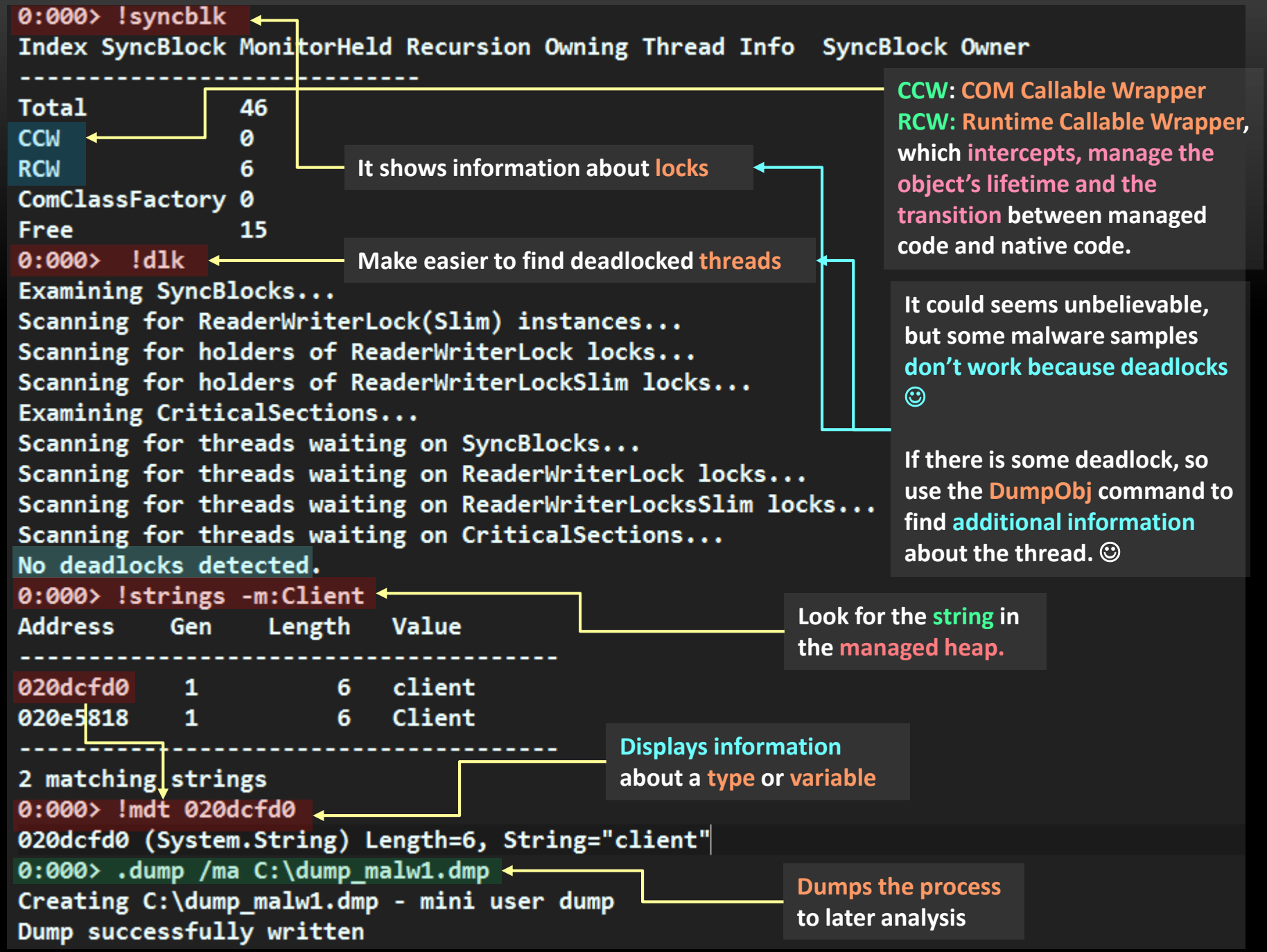
```
0:000> !FinalizeQueue
SyncBlocks to be cleaned up: 0
MTA Interfaces to be released: 0
STA Interfaces to be released: 0
---------------------------------
generation 0 has 524 finalizable objects (0566d5b0->0566dde0)
generation 1 has 122 finalizable objects (0566d3c8->0566d5b0)
generation 2 has 0 finalizable objects (0566d3c8->0566d3c8)
Ready for finalization 0 objects (0566dde0->0566dde0)
Statistics:
      MT    Count    TotalSize Class Name
6a8e2448        1           12 System.Management.IWbemClassObjectFreeThreaded
66a051e8        1           16 System.Windows.Forms.Control+FontHandleWrapper
00558164        1           16 testprint.KeyboardHook
6ab96998        1           20 Microsoft.Win32.SafeHandles.SafeFileMappingHandle
6ab96940        1           20 Microsoft.Win32.SafeHandles.SafeViewOfFileHandle
6ab90a48        1           20 Microsoft.Win32.SafeHandles.SafeFileHandle
686eb94c        1           20 System.Drawing.FontFamily
679a6b04        1           20 System.Net.SafeNativeOverlapped
679a67c4        1           20 System.Net.SafeInternetHandle
6753ddb0        1           20 Microsoft.Win32.SafeHandles.SafeFileMapViewHandle
6753dd58        1           20 Microsoft.Win32.SafeHandles.SafeFileMappingHandle
6753566c        1           20 System.Net.SafeLocalFree
68ffe728        1           24
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase+WinFormsAppContext
67528ca4        1           32 System.ComponentModel.Container
6ab83238        2           40 Microsoft.Win32.SafeHandles.SafePEFileHandle
6ab7a13c        2           40 Microsoft.Win32.SafeHandles.SafeTokenHandle
67531c90        2           40 System.Net.SafeCloseSocket+InnerSafeCloseSocket
686eb57c        1           44 System.Drawing.Font
66a00ec0        1           44 System.Windows.Forms.NativeWindow
67533204        2           56 System.Net.SafeCloseSocketAndEvent
6ab83e60        1           60 System.Runtime.Remoting.Contexts.Context
6a8e270c        1           60 System.Management.ManagementObject
686ebf78        2           80 System.Drawing.Icon
```

- ✓ The Finalization Queue contains objects with finalizers (Finalize( )).
- ✓ When an object in Finalization Queue becomes rootless, so the GC put it into the f-reachable queue, which are considered garbage (but alive).

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

```
0:000> !GCHandles
GC Handle Statistics:
Strong Handles: 52
Pinned Handles: 10
Async Pinned Handles: 0
Ref Count Handles: 0
Weak Long Handles: 24
Weak Short Handles: 41
Other Handles: 0
Statistics:
      MT    Count    TotalSize Class Name
6ab92698        1           12 System.Object
6ab93148        1           28 System.SharedStatics
67528ef8        1           32 Microsoft.Win32.NativeMethods+WndProc
67528d70        1           32 Microsoft.Win32.NativeMethods+ConHndlr
0055832c        1           32 testprint.KeyboardHook+KBDLLHookProc
6ab7dea4        2           40 System.Security.Policy.Evidence
6754375c        1           40 System.Diagnostics.BooleanSwitch
6ab83e60        1           60 System.Runtime.Remoting.Contexts.Context
6ab92dc0        1           72 System.ExecutionEngineException
6ab92d30        1           72 System.StackOverflowException
6ab92ca0        1           72 System.OutOfMemoryException
6753e5d4        2           80 System.Diagnostics.TraceSwitch
66a00ec0        2           88 System.Windows.Forms.NativeWindow
6ab9325c        1          100 System.AppDomain
67538e04        5          100 System.Net.TimerThread+TimerQueue
6ab92e50        2          144 System.Threading.ThreadAbortException
6752c58c        4          144 System.Net.Logging+NclTraceSource
669fcb64        3          144 System.Windows.Forms.Timer
675390f8        4          160 System.Diagnostics.SourceSwitch
6ab93fc8       10          240 System.Reflection.Assembly
66a07abc        6          312 System.Windows.Forms.Timer+TimerNativeWindow
6ab93050        7          392 System.Threading.Thread
66a021f0       13          416 System.Windows.Forms.NativeMethods+WndProc
6ab910f4        9          432 System.Reflection.Module
6ab94740       14          504 System.Security.PermissionSet
00557138        1          508 testprint.Form2
```

Excessive or long-time pinned handles can cause CLR heap fragmentation.


0:000> !syncblk
Index SyncBlock MonitorHeld Recursion Owning Thread Info SyncBlock Owner
Total 46
CCW 0
RCW 6
ComClassFactory 0
Free 15
0:000> !dlk
Examining SyncBlocks...
Scanning for ReaderWriterLock(Slim) instances...
Scanning for holders of ReaderWriterLock locks...
Scanning for holders of ReaderWriterLockSlim locks...
Examining CriticalSections...
Scanning for threads waiting on SyncBlocks...
Scanning for threads waiting on ReaderWriterLock locks...
Scanning for threads waiting on ReaderWriterLocksSlim locks...
Scanning for threads waiting on CriticalSections...
No deadlocks detected.
0:000> !strings -m:Client
Address Gen Length Value
020dcfd0 1 6 client
020e5818 1 6 Client
2 matching strings
0:000> !mdt 020dcfd0
020dcfd0 (System.String) Length=6, String="client"
0:000> .dump /ma C:\dump_malw1.dmp
Creating C:\dump_malw1.dmp - mini user dump
Dump successfully written
It shows information about locks
CCW: COM Callable Wrapper
RCW: Runtime Callable Wrapper, which intercepts, manage the object's lifetime and the transition between managed code and native code.
Make easier to find deadlocked threads
It could seems unbelievable, but some malware samples don't work because deadlocks ☺
If there is some deadlock, so use the DumpObj command to find additional information about the thread. ☺
Look for the string in the managed heap.
Displays information about a type or variable
Dumps the process to later analysis

```
0:008> !DumpStackObjects
OS Thread Id: 0x10fc (8)
ESP/REG  Object   Name
ecx      01da0b98 System.Management.WbemDefPath
0564ebf8 01d9f094 System.Threading.AutoResetEvent
0564ec0c 01d9f094 System.Threading.AutoResetEvent
0564ecbc 01d9f0ac Microsoft.Win32.SafeHandles.SafeWaitHandle
0564ece4 01da6460 System.Management.MTAHelper+MTARequest
0564ed1c 01d9f088 System.Object
0564ed3c 01d9f0d8 System.Threading.ThreadStart
0564f1f0 01d9f164 System.Threading.ExecutionContext
0564f1f4 01da0048 System.Threading.ContextCallback
0564f208 01d9f130 System.Threading.ThreadHelper
0564f210 01d9f164 System.Threading.ExecutionContext
0564f214 01d9f130 System.Threading.ThreadHelper
0564f2e4 01d9f144 System.Threading.ThreadStart
0564f470 01bc2f8c System.Security.Principal.WindowsPrincipal
0564f478 01d9f144 System.Threading.ThreadStart
0564f484 01d9f0f8 System.Threading.Thread
0564f48c 01d9f144 System.Threading.ThreadStart
0:008> !do 01d9f094
Name: System.Threading.AutoResetEvent
MethodTable: 6759de18
EEClass: 67365470
Size: 24(0x18) bytes
 (C:\Windows\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\mscorlib.dll)
Fields:
      MT    Field   Offset                 Type VT     Attr    Value Name
675b2698  400018a        4        System.Object  0 instance 00000000 __identity
675b5344  40005bf        c        System.IntPtr  1 instance     2f0 waitHandle
67599ab8  40005c0        8 ...es.SafeWaitHandle  0 instance 01d9f0ac safeWaitHandle
67586548  40005c1       10       System.Boolean  1 instance        0 hasThreadAffinity
675b5344  40005c2      9ac        System.IntPtr  1   shared   static InvalidHandle
    >> Domain:Value  003a5ba8:ffffffff <<
0:008> !handle 2f0 8
Handle 2f0
  Object Specific Information
    Event Type Auto Reset
    Event is Waiting
```

- ✓ Remember that an event works as synchronization object. ☺
- ✓ When an event happens (going from non-signaled state to signaled state), the waiting thread (WaitForSingleObject( )) starts its execution.
- ✓ Auto reset: If the event is signaled, so allows the thread being release and it is automatically reset to non-signaled state.
- ✓ Manual reset: the event remains in signaled state until being intentionally reset.
- ✓ Other synchonization techniques could be Semaphores, ReaderWriterLock, Mutex and so on...

It shows specific-object handle information

```
0:000> !teb
TEB at 7ffde000
    ExceptionList:        001df3cc
    StackBase:            001e0000
    StackLimit:           001d9000
    SubSystemTib:         00000000
    FiberData:            00001e00
    ArbitraryUserPointer: 00000000
    Self:                 7ffde000
    EnvironmentPointer:   00000000
    ClientId:             000002c0 . 00000f18
    RpcHandle:            00000000
    Tls Storage:          7ffde02c
    PEB Address:          7ffdf000
    LastErrorValue:       0
    LastStatusValue:      c0000034
    Count Owned Locks:    0
    HardErrorMode:        0

0:000> dps 001d9000 001e0000
001db0e0  68830306 MSVCR80!_woutput_s_l+0x8d9 [f:\dd\vctools\crt_bld\self_x86\crt\src\output.c @ 2377]
001db2e4  6a956564 mscorlib_ni!IndexOfKey+0x21 [f:\dd\ndp\clr\src\BCL\System\Collections\SortedList.cs @ 446]
001db2f4  6a9516f4 mscorlib_ni!MoveNext+0x2b [f:\dd\ndp\clr\src\BCL\System\Collections\ArrayList.cs @ 1943]
001db318  6a921000 mscorlib_ni!FindAll+0x47 [f:\dd\ndp\clr\src\BCL\System\Array.cs @ 849]
001db3d4  68832592 MSVCR80!_fputwc_nolock+0x1dc [f:\dd\vctools\crt_bld\self_x86\crt\src\fputwc.c @ 154]
001db3d8  68822e24 MSVCR80!_getptd_noexit+0x72 [f:\dd\vctools\crt_bld\self_x86\crt\src\tidtable.c @ 633]
001db3ec  6882f101 MSVCR80!write_char+0x16 [f:\dd\vctools\crt_bld\self_x86\crt\src\output.c @ 2440]
[f:\dd\ndp\fx\src\Regex\System\Text\RegularExpressions\RegexBoyerMoore.cs @ 60]
001dfab8  6a967140 mscorlib_ni!get_ParameterType+0x19 [f:\dd\ndp\clr\src\BCL\System\Reflection\XXXInfos.cs
@ 3850]
001dfe40  6b55360c mscorwks!ExecuteEXE+0x59
001dfe44  011f0000 MicroTik_Realtek_Driver
001dfe90  6b55353c mscorwks!_CorExeMain+0x15c
001dfe94  011f0000 MicroTik_Realtek_Driver
001dfea0  6b5534a4 mscorwks!_CorExeMain
```

Additionally, it is always recommended to investigate the current stack, looking for some interesting string ☺

Few hints about our malware...☺

```
0:000> !DumpHeap
...
Statistics:
      MT    Count    TotalSize Class Name
005565f4        1           12
testprint.My.MyProject+ThreadSafeObjectProvider`1[[testprint.My.MyProject+MyWebServices, MicroTik Realtek Driver]]
005564e4        1           12 testprint.My.MyProject+ThreadSafeObjectProvider`1[[testprint.My.MyProject+MyForms,
MicroTik Realtek Driver]]
0055634c        1           12
testprint.My.MyProject+ThreadSafeObjectProvider`1[[Microsoft.VisualBasic.ApplicationServices.User,
Microsoft.VisualBasic]]
005562ec        1           12 testprint.My.MyProject+ThreadSafeObjectProvider`1[[testprint.My.MyApplication,
MicroTik Realtek Driver]]
005561cc        1           12 testprint.My.MyProject+ThreadSafeObjectProvider`1[[testprint.My.MyComputer, MicroTik
Realtek Driver]]
00558164        1           16 testprint.KeyboardHook
00556494        1           20 testprint.My.MyProject+MyForms
00557bb4        1           24 testprint.cTripleDES
00558004        1           28 testprint.UploadScreenshot
00558410        1           32 testprint.KeyboardHook+KeyDownEventHandler
0055832c        1           32 testprint.KeyboardHook+KBDLLHookProc
0055820c        1           32 testprint.Client+NeuerTextIstDaEventHandler
005560f4        1           32 testprint.My.MyComputer
00557cf0        1           44 testprint.Client
00553f3c        1          108 testprint.My.MyApplication,
00557138        1          508 testprint.Form2
0:000> !DumpHeap -type testprint.KeyboardHook
 Address       MT     Size
02057fb4 00558164       16
02057fc4 0055832c       32
0205808c 00558410       32
total 3 objects
Statistics:
      MT    Count    TotalSize Class Name
00558164        1           16 testprint.KeyboardHook
00558410        1           32 testprint.KeyboardHook+KeyDownEventHandler
0055832c        1           32 testprint.KeyboardHook+KBDLLHookProc
Total 3 objects
```

Get objects (and their respective metadata) stored in the heap. To a short output, use !DumpHeap -stat

Dumps the heap, but limit the output to the specified type name.

Class !

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

```
0:000> !address 02057fb4
Usage:                  <unclassified>
Allocation Base:        02050000
Base Address:           02050000
End Address:            02412000
Region Size:            003c2000
Type:                   00020000    MEM_PRIVATE
State:                  00001000    MEM_COMMIT
Protect:                00000004    PAGE_READWRITE

0:000> !dumpmt -md 00558164
EEClass: 00654fac
Module: 00552c6c
Name: testprint.KeyboardHook
mdToken: 0200000d  (C:\malware1.exe)
BaseSize: 0x10
ComponentSize: 0x0
Number of IFaces in IFaceMap: 0
Slots in VTable: 13
--------------------------------------
MethodDesc Table
   Entry MethodDesc      JIT Name
6aae7ae0   6a9649d8   PreJIT System.Object.ToString()
6aae7b00   6a9649e0   PreJIT System.Object.Equals(System.Object)
6aae7b70   6a964a10   PreJIT System.Object.GetHashCode()
0055c61d   00558138     NONE testprint.KeyboardHook.Finalize()
006908f0   00558130      JIT testprint.KeyboardHook..ctor()
0055c6b8   00558088     NONE testprint.KeyboardHook.SetWindowsHookEx(Int32, KBDLLHookProc, IntPtr, Int32)
0055c5fd   005580ac     NONE testprint.KeyboardHook.CallNextHookEx(Int32, Int32, IntPtr, IntPtr)
0055c601   005580d0     NONE testprint.KeyboardHook.UnhookWindowsHookEx(Int32)
00690a20   005580f4      JIT testprint.KeyboardHook.add_KeyDown(KeyDownEventHandler)
0055c609   00558100     NONE testprint.KeyboardHook.remove_KeyDown(KeyDownEventHandler)
0055c60d   0055810c     NONE testprint.KeyboardHook.add_KeyUp(KeyUpEventHandler)
0055c611   00558118     NONE testprint.KeyboardHook.remove_KeyUp(KeyUpEventHandler)
0055c615   00558124     NONE testprint.KeyboardHook.KeyboardProc(Int32, IntPtr, IntPtr)
```

Displays information about the method table

```
0:000> !DumpIL 0x558124
ilAddr = 011fab24
IL_0000: ldarg.1
IL_0001: brtrue IL_00cc
IL_0006: ldarg.2
IL_0007: stloc.1
IL_0008: ldloc.1
IL_0009: ldc.i4 256
IL_000e: call System.IntPtr::op_Explicit
IL_0013: call System.IntPtr::op_Equality
IL_0018: brtrue.s IL_002c
IL_001a: ldloc.1
IL_001b: ldc.i4 260
IL_0020: call System.IntPtr::op_Explicit
IL_0025: call System.IntPtr::op_Equality
IL_002a: brfalse.s IL_006b
IL_002c: ldsfld testprint.KeyboardHook::KeyDownEvent
IL_0031: stloc.2
IL_0032: ldloc.2
IL_0033: brfalse IL_00cc
IL_0038: ldloc.2
IL_0039: ldarg.3
IL_003a: ldloc.0
IL_003b: box KBDLLHOOKSTRUCT
IL_0040: call System.Object::GetType
IL_0045: call System.Runtime.InteropServices.Marshal::PtrToStructure
IL_004a: dup
IL_004b: brtrue.s IL_0059
IL_004d: pop
IL_004e: ldloca.s VAR OR ARG 3
IL_0050: initobj KBDLLHOOKSTRUCT
IL_0056: ldloc.3
IL_0057: br.s IL_005e
IL_0059: unbox.any KBDLLHOOKSTRUCT
IL_005e: ldfld KBDLLHOOKSTRUCT::vkCode
```

!DumpIL displays the IL instructions of a method

Boxing turns a value type into an object reference (reference type)

Unboxing turns a object reference into a value type

```
0:000> .shell -ci "!DumpHeap -strings" findstr /i http://
       1           64 "http://postocentro.com"
       2           64 "http://"
       1          104 "http://www.w3.org/2000/09/xmldsig# X509Data"
       1          104 "http://www.w3.org/2000/09/xmldsig# KeyName"
       1          108 "http://www.w3.org/TR/1999/REC-xpath-19991116"
       1          108 "http://www.w3.org/2001/04/xmlenc#kw-tripledes"
       1          108 "http://www.w3.org/2001/04/xmldsig-more#sha384"
       1          108 "http://postocentro.com/CL/PostaEssaPORRA.php?"
       1          120 "http://www.w3.org/2000/09/xmldsig# RetrievalMethod"
       1          120 "http://schemas.microsoft.com/.NetConfiguration/v2.0"
       1          124 "http://www.w3.org/2001/04/xmldsig-more#hmac-ripemd160"
       1          124 "http://www.w3.org/2000/09/xmldsig#enveloped-signature"
       1          128 "http://www.w3.org/2000/09/xmldsig# KeyValue/RSAKeyValue"
       1          128 "http://www.w3.org/2000/09/xmldsig# KeyValue/DSAKeyValue"
       1          140 "http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"
       1          184 "        Dim strarg As string ="http://postocentro.com/CL/PostaE"
       2          652 "http://postocentro.com/CL/PostaEssaPORRA.php?plug=SIM&GBS=Desco"

0:000> .shell -ci "!DumpHeap -strings" findstr /i posto
       1           64 "http://postocentro.com"
       2           96 "postocentro.com"
       1          108 "http://postocentro.com/CL/PostaEssaPORRA.php?"
       1          132 "The remote name could not be resolved: 'postocentro.com'"
       1          184 "        Dim strarg As string ="http://postocentro.com/CL/PostaE"
       2          652 "http://postocentro.com/CL/PostaEssaPORRA.php?plug=SIM&GBS=Desco"

0:000> .shell -ci "!DumpHeap -strings" findstr /i banco
       4          144 "BANCOITA?"
       4          160 "BANCOESTADO"
       4          176 "BANCOSECURITY"
       4          176 "BANCODECHILE"
       4          192 "BANCOSANTANDER"
.shell: Process exited

0:000> .shell -ci "!DumpHeap -strings" findstr /i itau
       7          336 "itauaplicativo"
.shell: Process exited
```

!DumpHeap -strings is always excellent to find valuable strings. ☺

If you can't recognise these strings, they are related to banks. ☺

- ✓ Other possible WinDbg breakpoints that could be used to gather further information:

- ✓ How to log API calls:
  - ✓ bp mscorwks!MethodTable::MapMethodDeclToMethodImpl
  - ✓ bp clr!MethodTable::MapMethodDeclToMethodImpl

- ✓ How to get possible strings:
  - ✓ !bpmd mscorlib.dll System.String.CreateStringFromEncoding
  - ✓ !bpmd mscorlib.dll System.String.Intern
  - ✓ !bpmd mscorlib.dll System.Text.StringBuilder.ToString
  - ✓ bp mscorwks!GlobalStringLiteralMap::GetStringLiteral
  - ✓ bp clr!StringLiteralMap::GetstringLiteral

- ✓ How to examine loaded assemblies:
  - ✓ bp mscorwks!CLRMapViewOfFileEx
  - ✓ bp clr!AssemblyNative::LoadFromBuffer

```
C:\>python malwoverview\malwoverview.py -u http://postocentro.com -w 1 -b 1

URL SUMMARY REPORT
--------------------

Status:           Scan request successfully queued, come back later for the report
Scan date:        2019-07-23 06:46:16
Scan ID:          41fe5492405e3adc488fb476a33d4e9db0f16fee5c1b6449d3e2f81a71413c5e-1563864376
URL:              http://postocentro.com/
Permanent Link:   https://www.virustotal.com/url/41fe5492405e3adc488fb476a33d4e9db0f16fee5c1b6449d3e2f81a71413c5e/analysis/1563864376/
Result VT:        3/70

URL DETAILED REPORT
--------------------

AlienVault:       clean site
Avira:            clean site
BitDefender:      clean site
CyRadar:          clean site
ESET:             malware site
Forcepoint:       clean site
Fortinet:         clean site
G-Data:           clean site
Google:           clean site
Kaspersky:        malware site
Malc0de:          clean site
MalwarePatrol:    clean site
OpenPhish:        clean site
PhishLabs:        unrated site
Phishtank:        clean site
Sophos:           malicious site
Trustwave:        clean site
VX Vault:         clean site
ZeroCERT:         clean site
```

Surprise... is it malicious? ☺

❑ https://github.com/alexandreborges/malwoverview

```
C:\malwoverview>python malwoverview.py -r postocentro.com -w 1 -b 1

DOMAIN SUMMARY REPORT
---------------------

Undetected Referrer Samples:
Detected Referrer Samples:

Whois Timestamp:
                        2019-01-29 18:30:22

Undetected Downld. Samples:
                        date: 2018-11-06 05:51:04
                        total: 56
                        sha256: f1945cd6c19e56b3c1c78943ef5ec18116907a4ca1efc40a57d48ab1db7adfc5

Detected Downloaded Samples:
Resolutions:
                        last resolved: 2019-01-03 08:30:14
                        ip address:    160.153.50.64

                        last resolved: 2019-02-13 23:11:20
                        ip address:    184.168.221.64

                        last resolved: 2019-02-01 18:40:21
                        ip address:    50.63.202.76

                        last resolved: 2019-01-11 02:06:28
                        ip address:    50.63.202.91

Subdomains:
                        www.postocentro.com
                        span.postocentro.com
```

Categories:

blogs
web hosting

Domain Siblings:

Detected URLs:

url: http://postocentro.com/
positives: 3
total: 70
scan_date: 2019-07-23 06:46:16

url: http://postocentro.com/Whats/DBWhatsApp2018.php
positives: 4
total: 66
scan_date: 2019-03-20 00:42:20

url: http://postocentro.com/GUESS/PostBabyUSXPK2.phpplug=&GBS=&SYS=&USERPC=&AVS=&NAV=&ORI=1.1DPOSTapplication/x-www-form-urlencodedCould
positives: 5
total: 67
scan_date: 2019-02-22 10:42:32

url: http://postocentro.com/error
positives: 5
total: 67
scan_date: 2018-12-20 00:29:48

url: http://postocentro.com/iBP/index20.php
positives: 4
total: 70
scan_date: 2018-11-13 07:25:11

url: http://postocentro.com/CAMPx2/Postaessecaralhofdp.php
positives: 4
total: 67
scan_date: 2018-11-06 05:51:01

❖ Acknowledgments to:

✓DEF CON staff, who have been always very kind with me.

✓You, who have reserved some time attend my talk.

✓Security is like a drunk: while walking back-and-forth, he always proceeds halfway through the remaining distance, but he never gets there. ☺

✓Remember: the best of this life are people. ☺

- ✓ Malware and Security Researcher.
- ✓ Speaker at DEF CON USA 2018
- ✓ Speaker at DEF CON China 2019
- ✓ Speaker at CONFidence Conference 2019 (Poland)
- ✓ Speaker at HITB 2019 Amsterdam
- ✓ Speaker at BSIDES 2019/2018/2017/2016
- ✓ Speaker at H2HC 2016/2015
- ✓ Speaker at BHACK 2018
- ✓ Consultant, Instructor and Speaker on Malware Analysis, Memory Analysis, Digital Forensics and Rootkits.
- ✓ Reviewer member of the The Journal of Digital Forensics, Security and Law.
- ✓ Referee on Digital Investigation: The International Journal of Digital Forensics & Incident Response

THANK YOU FOR ATTENDING MY TALK. ☺

- Twitter:

  @ale_sp_brazil
  @blackstormsecbr

- Website: http://www.blackstormsecurity.com
- LinkedIn: http://www.linkedin.com/in/aleborges
- E-mail: alexandreborges@blackstormsecurity.com