Unpacking for Dummies

31st Annual FIRST Conference

About Us

Paul Jung

@_ _thanat0s_ _

Rémi Chipaux,

@futex90

X86 aware anyone ??

Are you ready ?

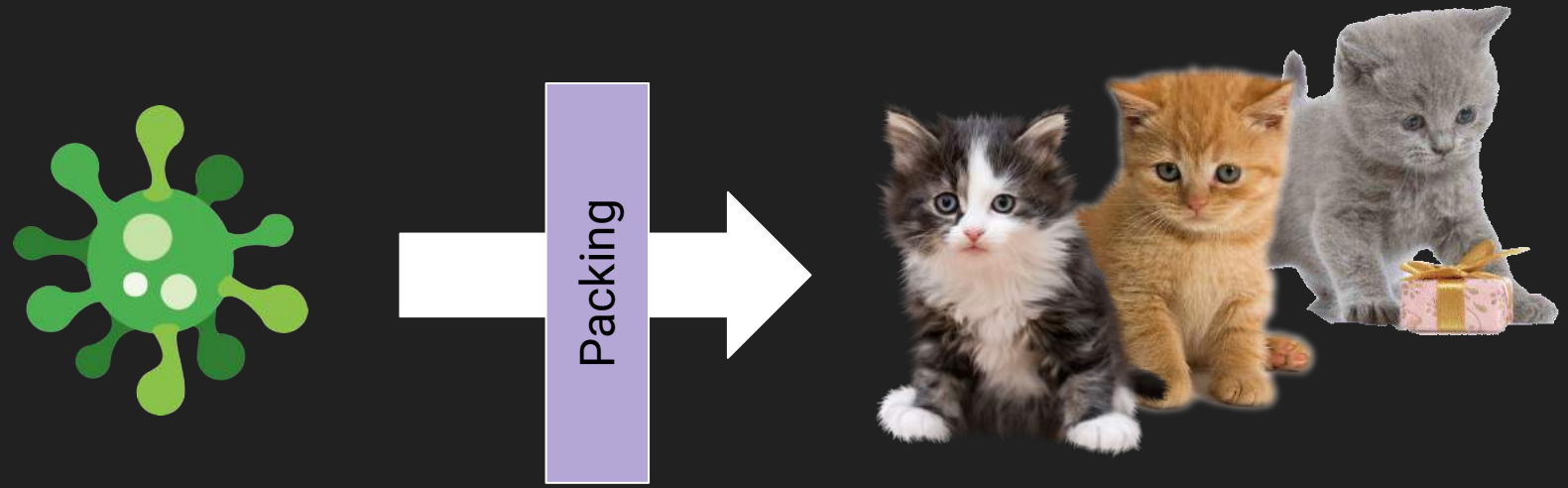
- **VM available online :**
  - http://upload.trollprod.org/Unpacking_Workshop_VmWare.zip
- **VM (vmware) from USB keys**

the password is : “reverse”

Why Packers

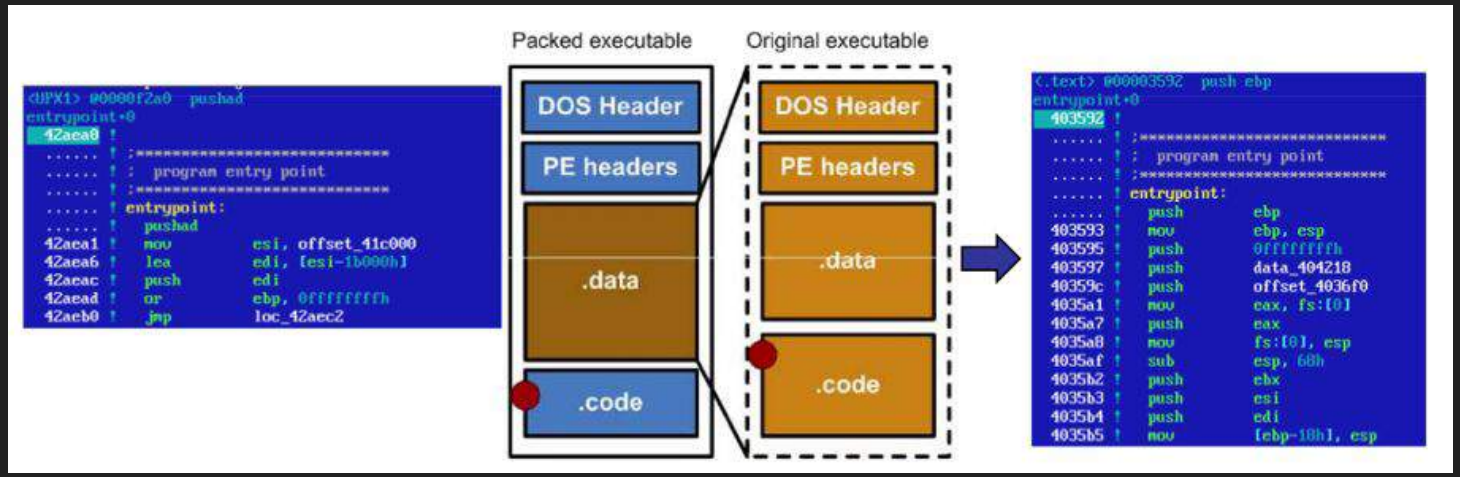
What is a Packer

- You may name it packer, cryptor or protector
- Convert a single executable into “army” of executable
- You may see it as a kind of matrioska



Why packers

- To avoid AV detection
- Get more time during the infection campaign
- Obfuscate globally the payload



Why un-packing

- After unpacking:
  - Identification of the real threat might be possible
- If still unknown:
  - You can reverse the unpacked sample

Why un-packing

- If successful:
  - Dynamic analysis of sample becomes possible

What kind of tools people use to pack

- Known tools/packer (upx, petite)
- Known “pro” packer (themida, vmprotect, ...)
- Dirty things, Self Extracting tools (SFX Cabs, Msi)
- Mostly, unknown packer/cryptor (??) …

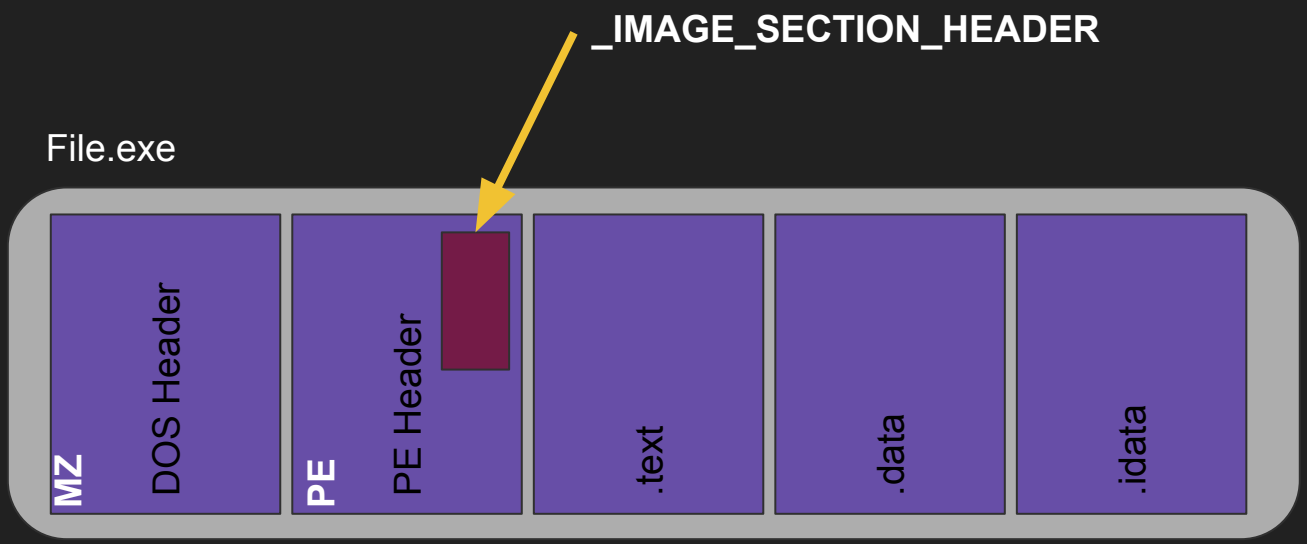
Concepts Needed

Mandatory to no leave the room in 10 minutes

Things to Know

- Mapping File to Memory
- Entry Point
- Import table
- Process Environment Block
- Traversing module list

Entry Point & File Mapping

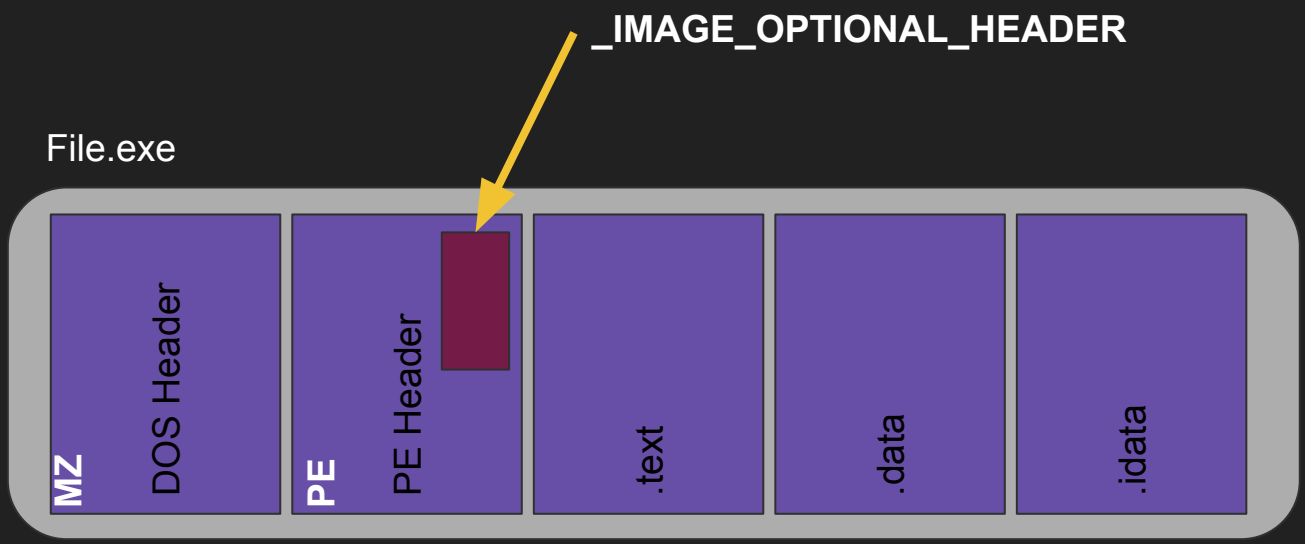


Sections

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000008cc  00401000  00401000  00000400  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         0000002c  00402000  00402000  00000e00  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .rdata        00000048  00403000  00403000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .bss          000001f4  00404000  00404000  00000000  2**2
                  ALLOC
  4 .idata        000002e0  00405000  00405000  00001200  2**2
                  CONTENTS, ALLOC, LOAD, DATA
SYMBOL TABLE:
no symbols
```

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Entry Point

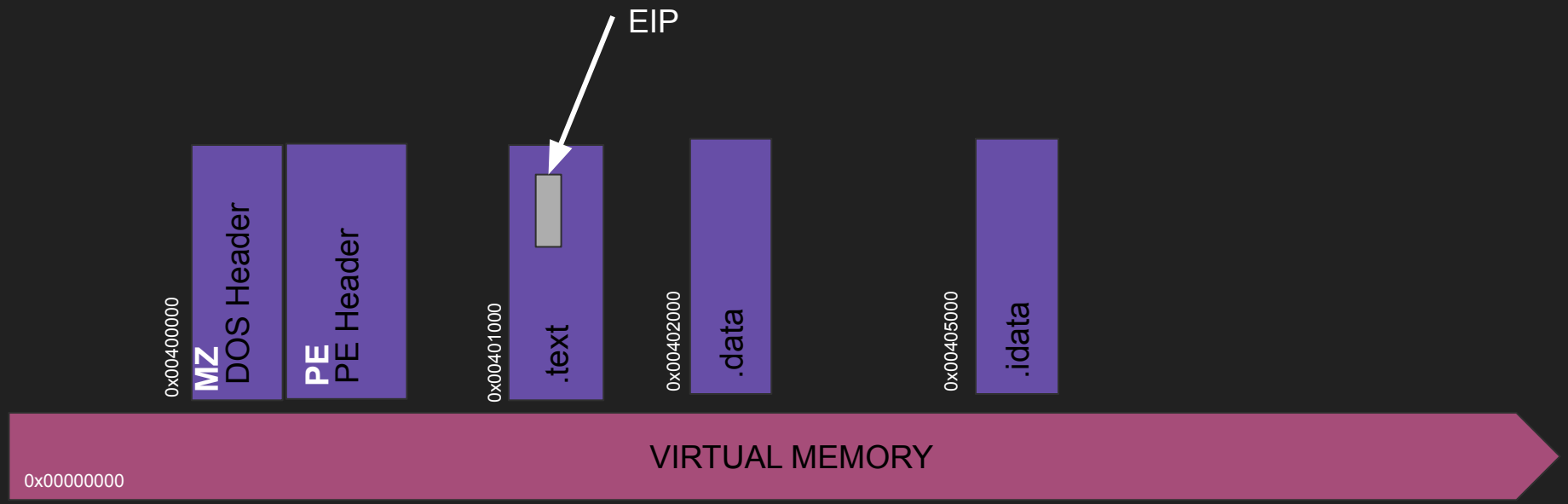

_IMAGE_OPTIONAL_HEADER
File.exe
MZ
DOS Header
PE
PE Header
.text
.data
.idata

Entry Point

```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD                 Magic;
  BYTE                 MajorLinkerVersion;
  BYTE                 MinorLinkerVersion;
  DWORD                SizeOfCode;
  DWORD                SizeOfInitializedData;
  DWORD                SizeOfUninitializedData;
  DWORD                AddressOfEntryPoint;
  DWORD                BaseOfCode;
  DWORD                BaseOfData;
  DWORD                ImageBase;
  DWORD                SectionAlignment;
  DWORD                FileAlignment;
  WORD                 MajorOperatingSystemVersion;
  WORD                 MinorOperatingSystemVersion;
  WORD                 MajorImageVersion;
  WORD                 MinorImageVersion;
  WORD                 MajorSubsystemVersion;
  WORD                 MinorSubsystemVersion;
  . . .
  DWORD                NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32
```

```
Time/Date                Fri Jul  4 10:34:06 2014
Magic                    010b    (PE32)
MajorLinkerVersion       2
MinorLinkerVersion       22
SizeOfCode               00000a00
SizeOfInitializedData    00000800
SizeOfUninitializedData  00000200
AddressOfEntryPoint      00001130
BaseOfCode               00001000
BaseOfData               00002000
ImageBase                00400000
SectionAlignment         00001000
FileAlignment            00000200
MajorOSystemVersion      4
MinorOSystemVersion      0
MajorImageVersion        1
MinorImageVersion        0
MajorSubsystemVersion    4
MinorSubsystemVersion    0
Win32Version             00000000
SizeOfImage              00006000
SizeOfHeaders            00000400
CheckSum                 0000b333
Subsystem                00000003        (Windows CUI)
DllCharacteristics       00000000
SizeOfStackReserve       00200000
SizeOfStackCommit        00001000
SizeOfHeapReserve        00100000
SizeOfHeapCommit         00001000
LoaderFlags              00000000
NumberOfRvaAndSizes      00000010
```

File Mapping



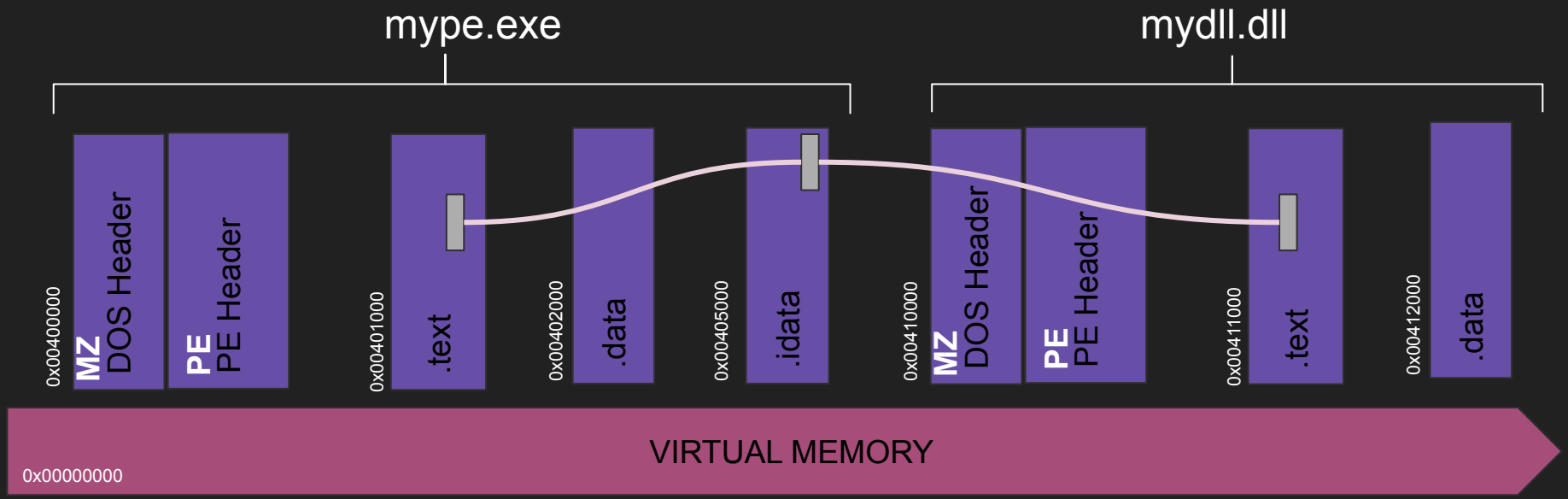
Import table

Import table list required functions for the PE.

A DLL is a PE

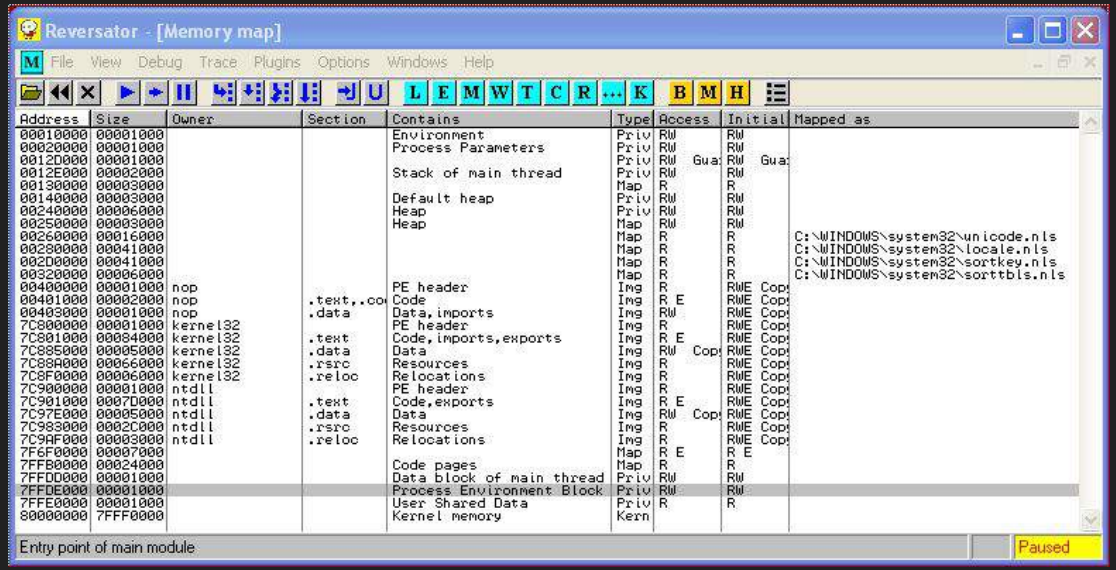
```
The Import Tables (interpreted .idata section contents)
 vma:            Hint    Time      Forward  DLL       First
                 Table   Stamp     Chain    Name      Thunk
 00005000       00005054 00000000 00000000 00005278 000050c0

        DLL Name: KERNEL32.dll
        vma:  Hint/Ord Member-Name Bound-To
        5128       156  ExitProcess
        5138       337  GetModuleHandleA
        514c       364  GetProcAddress
        5160       739  SetUnhandledExceptionFilter
        5180       751  Sleep

 00005014       00005070 00000000 00000000 000052b8 000050dc

        DLL Name: msvcrt.dll
        vma:  Hint/Ord Member-Name Bound-To
        5188        39  __getmainargs
        5198        60  __p__environ
        51a8        62  __p__fmode
        51b8        80  __set_app_type
        51cc       121  _cexit
        51d8       233  _iob
        51e0       350  _onexit
        51ec       388  _setmode
        51f8       540  atexit
        5204       642  puts
        520c       656  signal
        5218       659  sprintf

 00005028       000050a8 00000000 00000000 000052d4 00005114

        DLL Name: WS2_32.DLL
        vma:  Hint/Ord Member-Name Bound-To
        5224        62  WSASocketA
        5234        64  WSAStartup
        5244        79  closesocket
        5254        84  gethostbyname
```

File Mapping

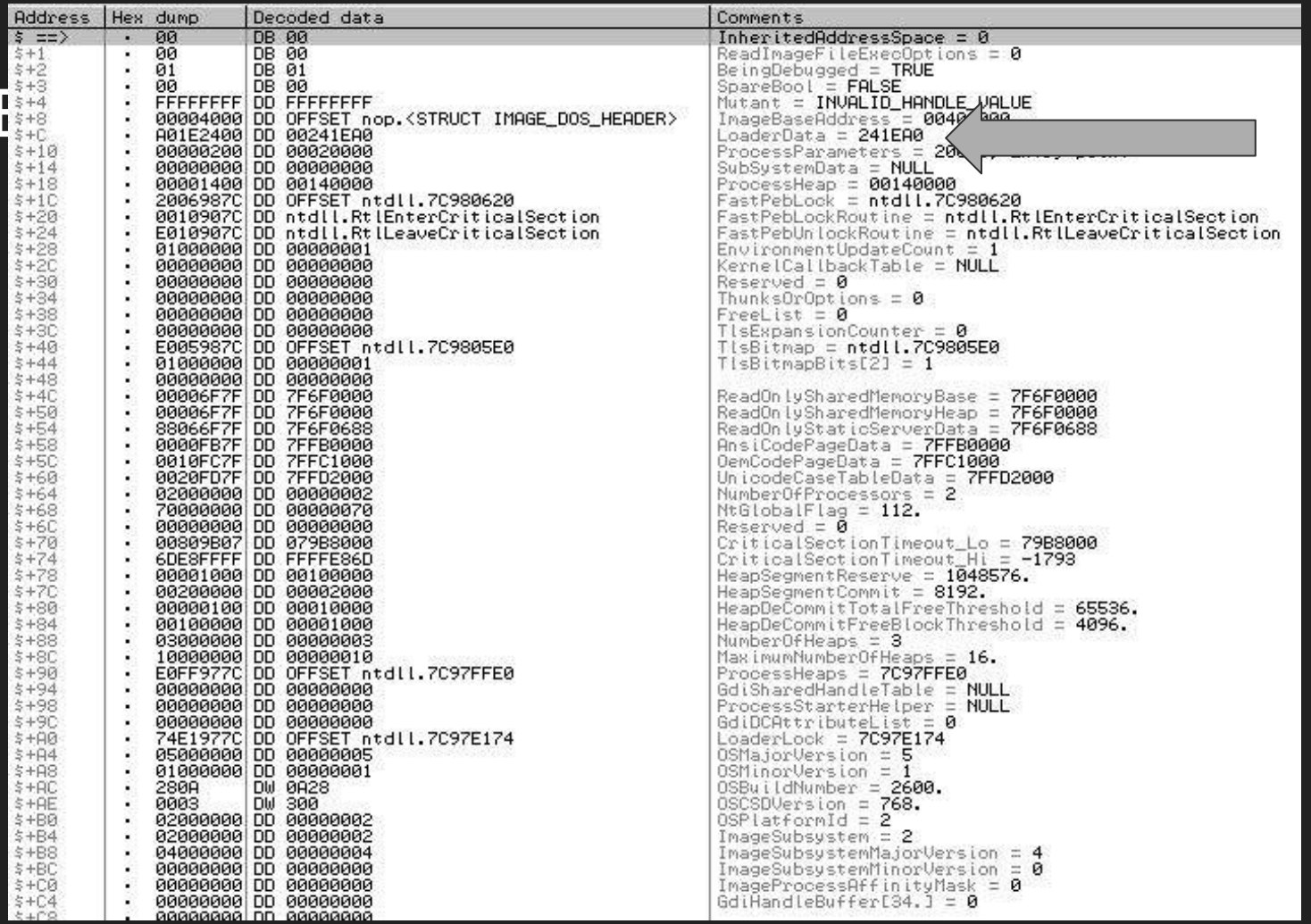


PEB (Process Environment Block)

- Memory structure with the process states
- Location
  - 32 Bits FS[0x30]
  - 64 Bits GS[0x60]



PEB



Address	Hex dump	Decoded data	Comments
$ ==>	00	DB 00	InheritedAddressSpace = 0
$+1	00	DB 00	ReadImageFileExecOptions = 0
$+2	01	DB 01	BeingDebugged = TRUE
$+3	00	DB 00	SpareBool = FALSE
$+4	FFFFFFFF	DD FFFFFFFF	Mutant = INVALID_HANDLE_VALUE
$+8	00004000	DD OFFSET nop.<STRUCT IMAGE_DOS_HEADER>	ImageBaseAddress = 0040
$+C	A01E2400	DD 00241EA0	LoaderData = 241EA0
$+10	00000200	DD 00020000	ProcessParameters = 20
$+14	00000000	DD 00000000	SubSystemData = NULL
$+18	00001400	DD 00140000	ProcessHeap = 00140000
$+1C	2006987C	DD OFFSET ntdll.7C980620	FastPebLock = ntdll.7C980620
$+20	0010907C	DD ntdll.RtlEnterCriticalSection	FastPebLockRoutine = ntdll.RtlEnterCriticalSection
$+24	E010907C	DD ntdll.RtlLeaveCriticalSection	FastPebUnlockRoutine = ntdll.RtlLeaveCriticalSection
$+28	01000000	DD 00000001	EnvironmentUpdateCount = 1
$+2C	00000000	DD 00000000	KernelCallbackTable = NULL
$+30	00000000	DD 00000000	Reserved = 0
$+34	00000000	DD 00000000	ThunksOrOptions = 0
$+38	00000000	DD 00000000	FreeList = 0
$+3C	00000000	DD 00000000	TlsExpansionCounter = 0
$+40	E005987C	DD OFFSET ntdll.7C9805E0	TlsBitmap = ntdll.7C9805E0
$+44	01000000	DD 00000001	TlsBitmapBits[2] = 1
$+48	00000000	DD 00000000	
$+4C	00006F7F	DD 7F6F0000	ReadOnlySharedMemoryBase = 7F6F0000
$+50	00006F7F	DD 7F6F0000	ReadOnlySharedMemoryHeap = 7F6F0000
$+54	88066F7F	DD 7F6F0688	ReadOnlyStaticServerData = 7F6F0688
$+58	0000FB7F	DD 7FFB0000	AnsiCodePageData = 7FFB0000
$+5C	0010FC7F	DD 7FFC1000	OemCodePageData = 7FFC1000
$+60	0020FD7F	DD 7FFD2000	UnicodeCaseTableData = 7FFD2000
$+64	02000000	DD 00000002	NumberOfProcessors = 2
$+68	70000000	DD 00000070	NtGlobalFlag = 112.
$+6C	00000000	DD 00000000	Reserved = 0
$+70	00809B07	DD 079B8000	CriticalSectionTimeout_Lo = 79B8000
$+74	6DE8FFFF	DD FFFFE86D	CriticalSectionTimeout_Hi = -1793
$+78	00001000	DD 00100000	HeapSegmentReserve = 1048576.
$+7C	00200000	DD 00002000	HeapSegmentCommit = 8192.
$+80	00000100	DD 00010000	HeapDeCommitTotalFreeThreshold = 65536.
$+84	00100000	DD 00001000	HeapDeCommitFreeBlockThreshold = 4096.
$+88	03000000	DD 00000003	NumberOfHeaps = 3
$+8C	10000000	DD 00000010	MaximumNumberOfHeaps = 16.
$+90	E0FF977C	DD OFFSET ntdll.7C97FFE0	ProcessHeaps = 7C97FFE0
$+94	00000000	DD 00000000	GdiSharedHandleTable = NULL
$+98	00000000	DD 00000000	ProcessStarterHelper = NULL
$+9C	00000000	DD 00000000	GdiDCAttributeList = 0
$+A0	74E1977C	DD OFFSET ntdll.7C97E174	LoaderLock = 7C97E174
$+A4	05000000	DD 00000005	OSMajorVersion = 5
$+A8	01000000	DD 00000001	OSMinorVersion = 1
$+AC	280A	DW 0A28	OSBuildNumber = 2600.
$+AE	0003	DW 300	OSCSDVersion = 768.
$+B0	02000000	DD 00000002	OSPlatformId = 2
$+B4	02000000	DD 00000002	ImageSubsystem = 2
$+B8	04000000	DD 00000004	ImageSubsystemMajorVersion = 4
$+BC	00000000	DD 00000000	ImageSubsystemMinorVersion = 0
$+C0	00000000	DD 00000000	ImageProcessAffinityMask = 0
$+C4	00000000	DD 00000000	GdiHandleBuffer[34.] = 0
$+C8	00000000	DD 00000000	

Traversing module list

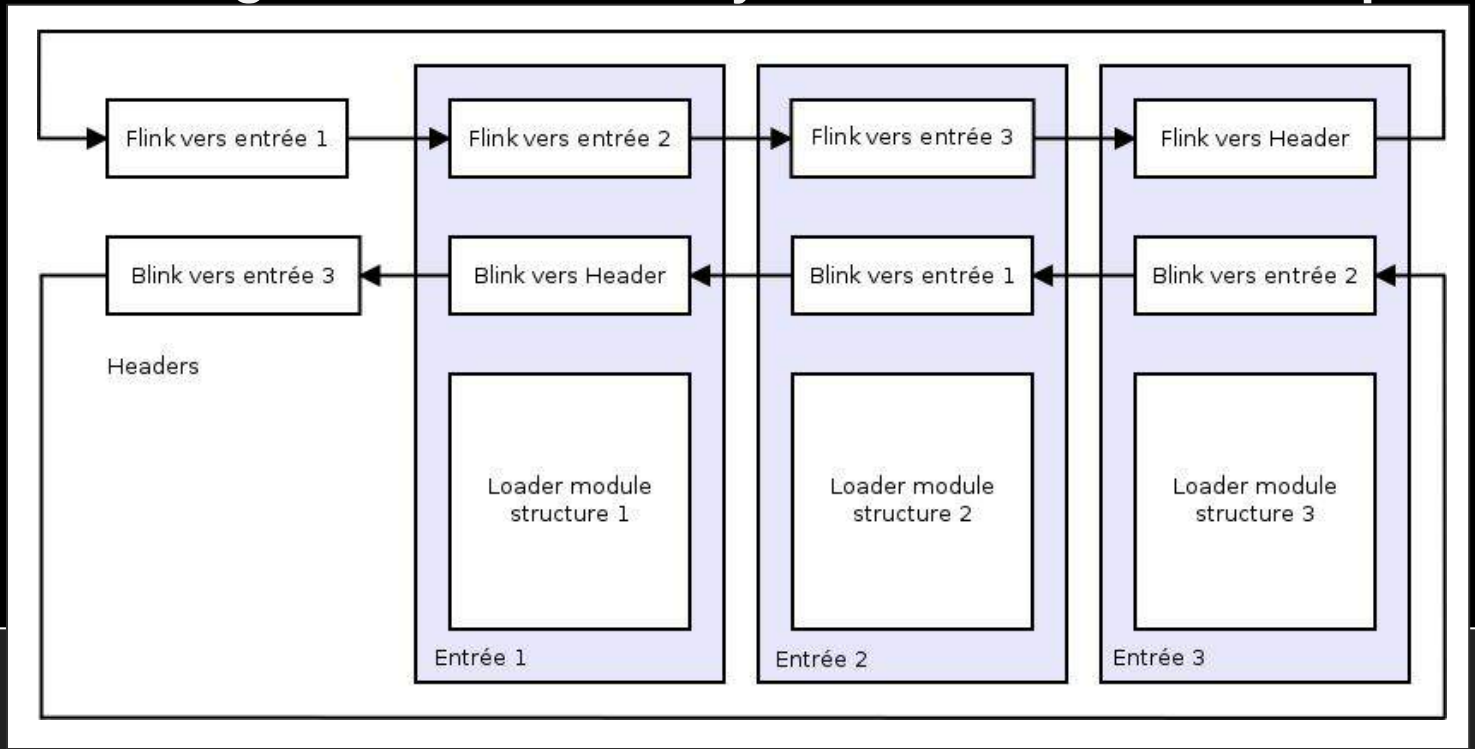
LoaderData gives DLL memory offset in the current process

3 Chained lists;

- **InLoadOrderModuleList**; DLL & PE at Start
- **InMemoryOrderModuleList**; DLL & PE, current state
- **InInitialisationOrderModuleList**; DLL loaded current state

Traversing module list

LoaderData gives DLL memory offset in the current process



Traversing module list

LoaderData gives DLL memory offset in the current process

```
push   30h
pop    ecx
mov    esi,fs:[ecx]   ; PEB (FS:[0x30])
mov    esi, [esi+0Ch]  ; ESI = LoaderData
mov    esi, [esi+1Ch]  ; ESI = Flink InInitialisationOrderModuleList
mov    ebp, [esi+8]    ; EBP = Base addresse de ntdll
mov    ds:ntdllbase, ebp
```

Traversing module list

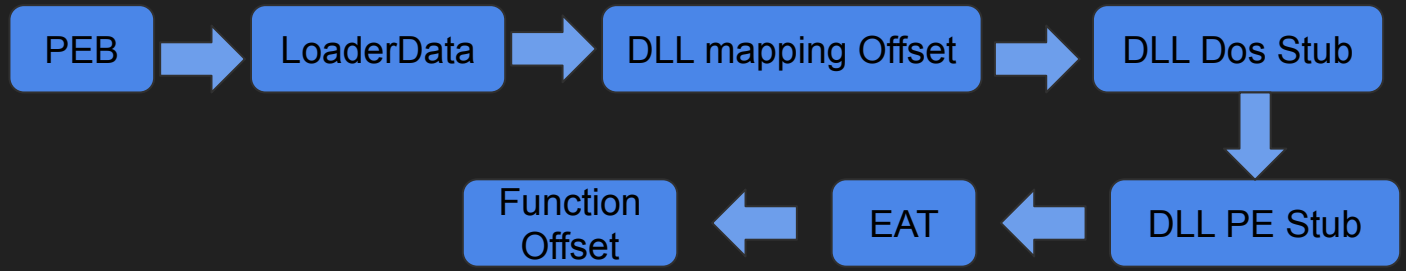
LoaderData gives DLL memory offset in the current process

- First one is always: **ntdll**
- Second one is always: **kernel32**

Traversing module list

LoaderData gives DLL memory offset in the current process

Parsing a PE (DLL) allows to find any function by hand.

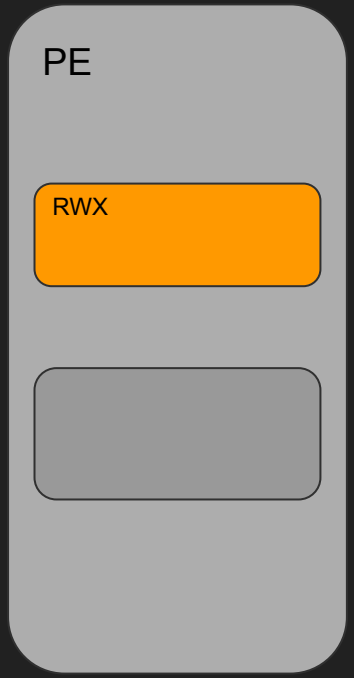


Packer families

How does it work

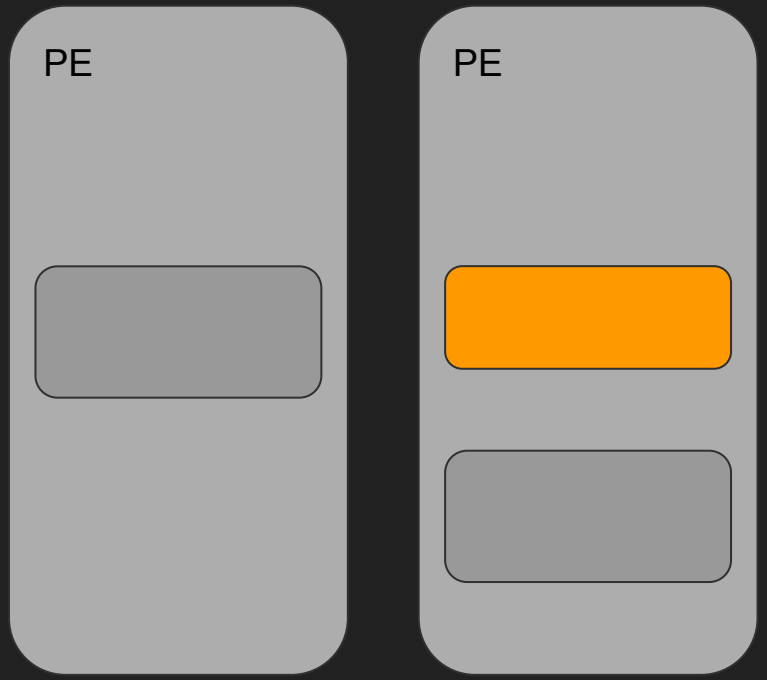
Mainly three kinds of techniques

- Unpack in the same process
  - Differents “flavors”
    - RWX native memory code segment in the PE:
      - Automodification of code,
      - Fix IAT,
      - Jump in it.
    - Allocate New RWX code segment:
      - Fill with code,
      - Fix IAT,
      - Jump in it.

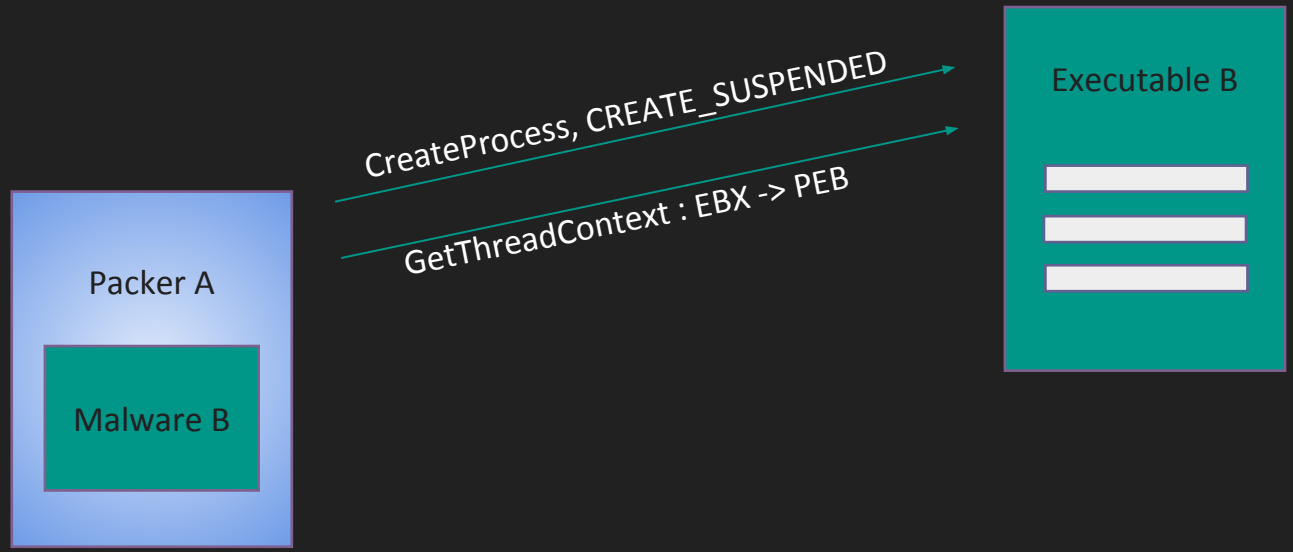


Mainly three kinds of techniques

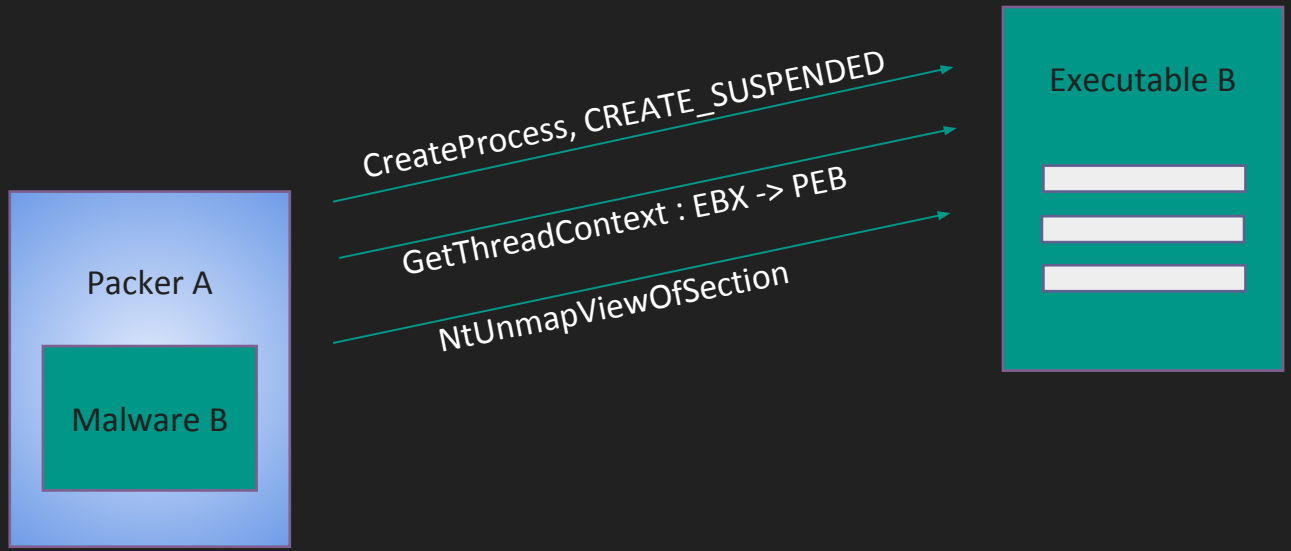
- Unpack in another process
  - Process hollowing aka RunPE
    - Create new “suspended” process
    - Unmap then replace all the segments
    - Set origin EIP
    - Release the Kraken !
    - exit



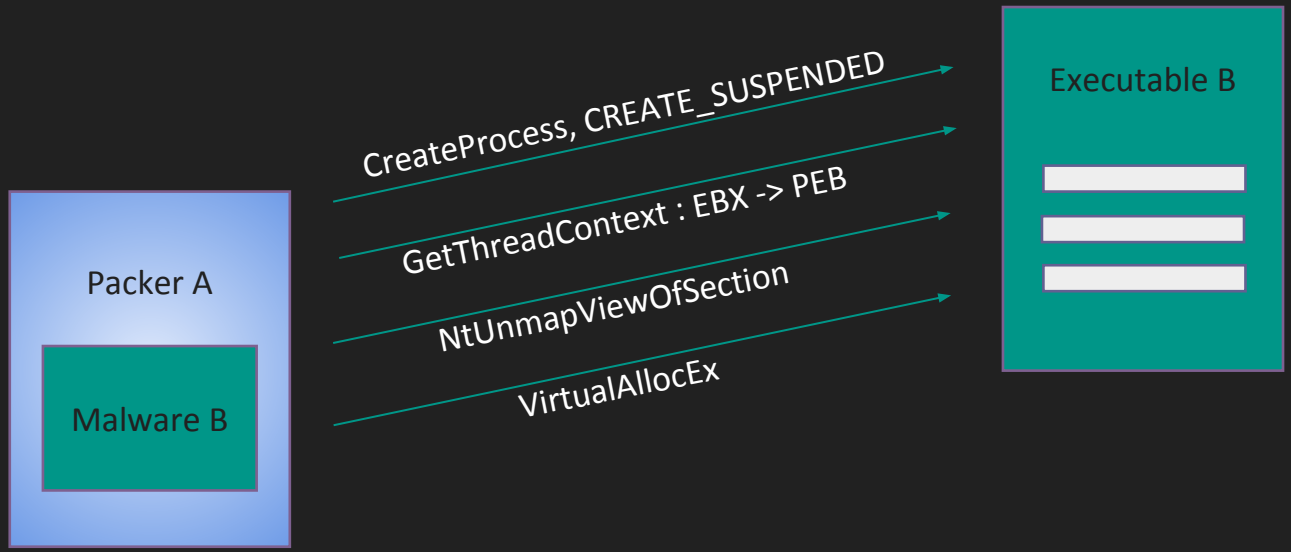
RunPE


Packer A
Malware B
CreateProcess, CREATE_SUSPENDED
GetThreadContext : EBX -> PEB
Executable B

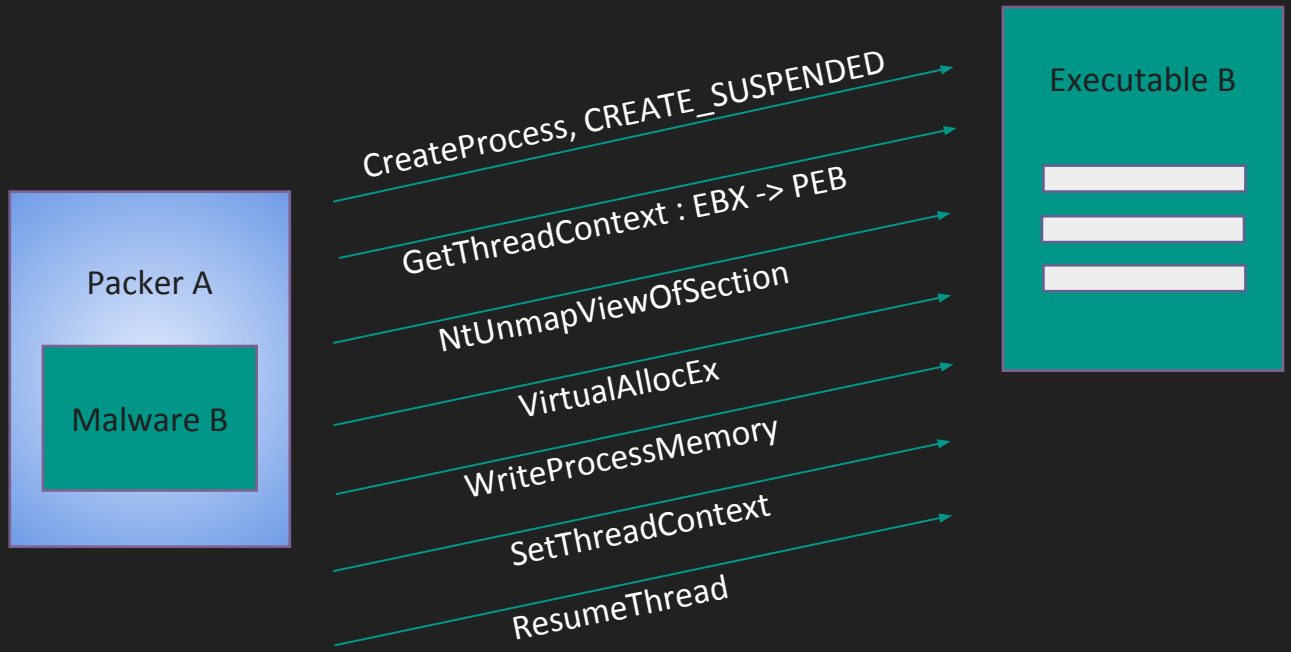
RunPE


Packer A
Malware B
CreateProcess, CREATE_SUSPENDED
GetThreadContext : EBX -> PEB
NtUnmapViewOfSection
Executable B

RunPE


Packer A
Malware B
CreateProcess, CREATE_SUSPENDED
GetThreadContext : EBX -> PEB
NtUnmapViewOfSection
VirtualAllocEx
Executable B

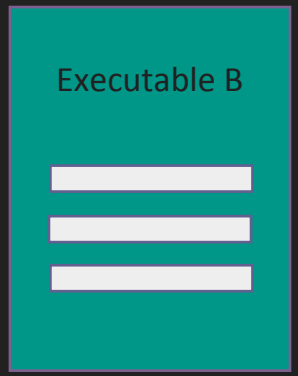
RunPE


Packer A
Malware B
CreateProcess, CREATE_SUSPENDED
GetThreadContext : EBX -> PEB
NtUnmapViewOfSection
VirtualAllocEx
WriteProcessMemory
SetThreadContext
ResumeThread
Executable B

RunPE

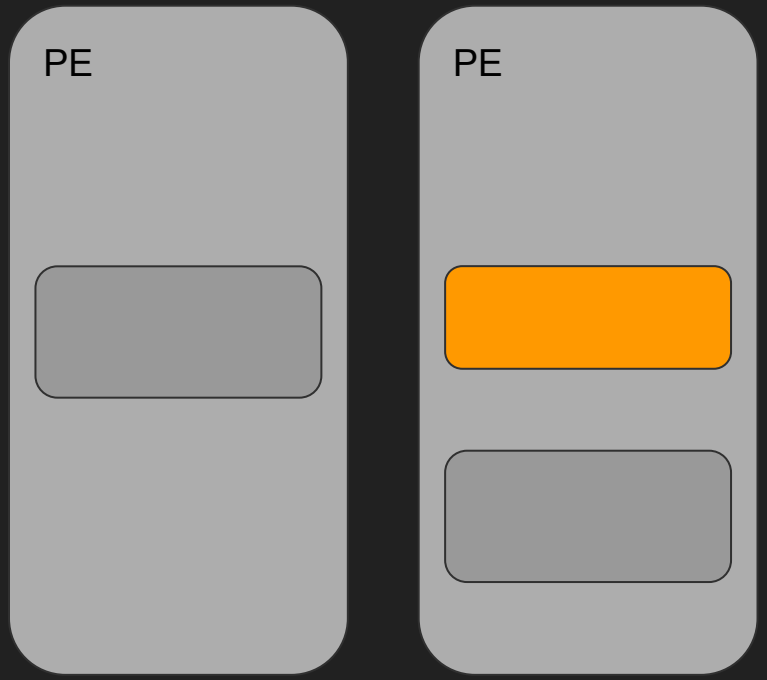
- Running executable is « Legit »
- No IAT fixing required

- Artefact
  - No parents



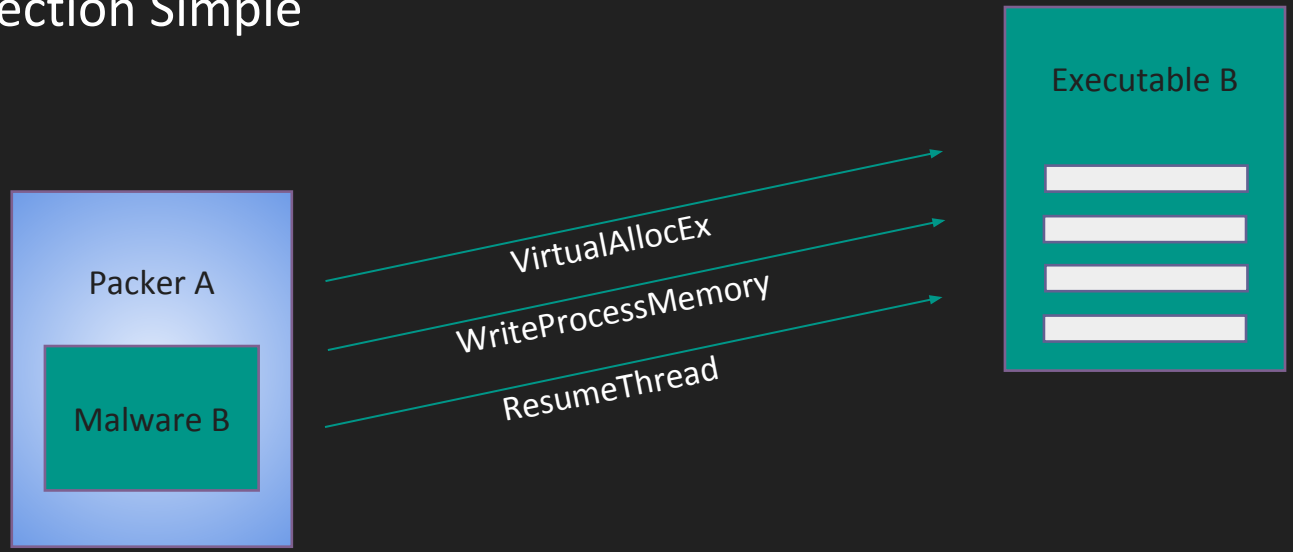
Mainly three kinds of techniques

- Unpack in another process
  - Create a new “thread” in another process
    - Create a section in a running process
    - Release the Kraken !
    - exit



Malware analysis

Injection Simple

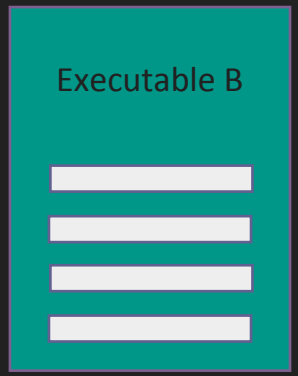


Malware analysis

Injection simple

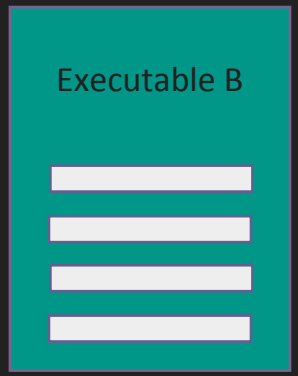
- Running executable is « Legit ».
- No IAT, direct function call required.

- Ends when Executable B is stopped.
  - Multiple injections usually



Malware analysis

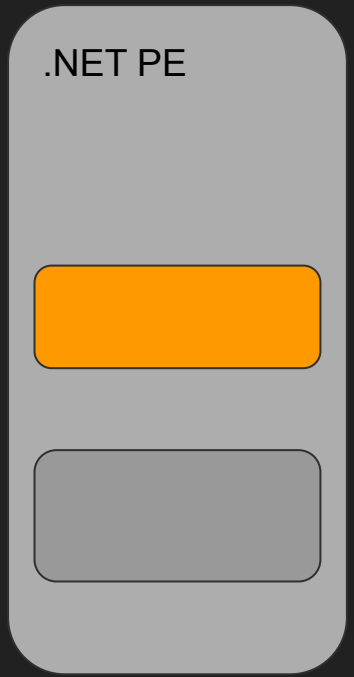
- **They are other techniques**
  - **Using CreatefileMapping, etc…**



But it’s enough for today !

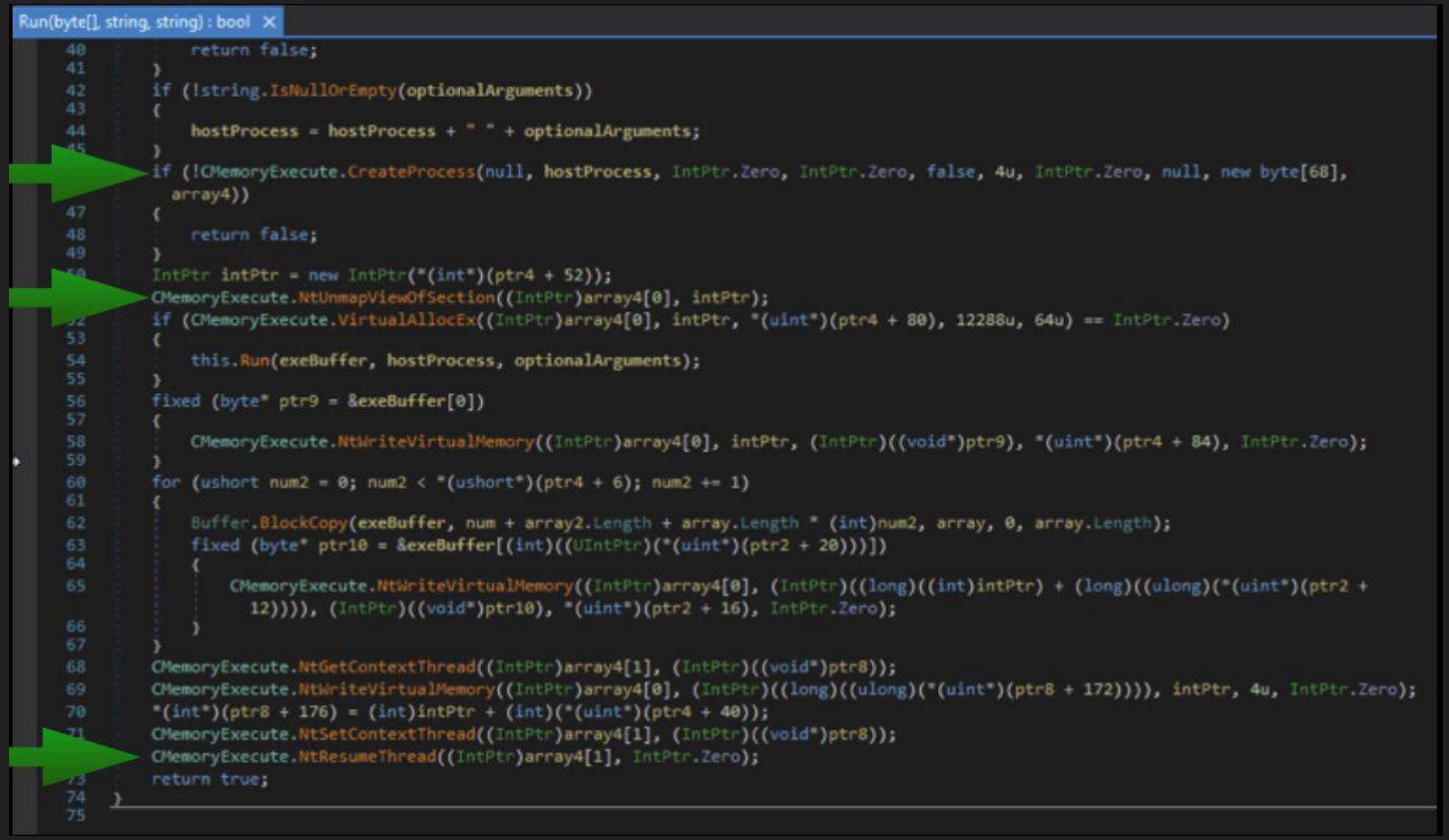
On .NET, many kind of techniques

- Load another module:
  - Sort of loading a “.NET DLL”
- Launch “Msil” code:
  - Using “assembly.invoke” directive
- Launch “Native” code:
  - Using “_ _asm {}”
- .NET based process hollowing:
  - Simple RunPE, launch another process



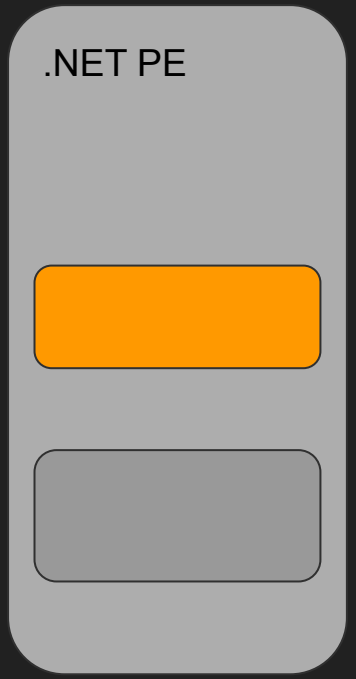
RunPE

Classical
RUNPE
In
.NET code



Where are the packed data ?

- Wherever it’s possible !
  - In a Data segment
  - In a code segment
  - In a ressource
- How ?
  - Xor, Aes, Base64, Bzip…
  - Or whatever it is possible to do
    - Who cares ?



Packer detection

How to know if it’s packed

Identifying that your sample is packed

A bunch of clues:

- **High section entropy (Above 6.5).. Maybe usual on ressources.**
- **Unusual small code segments.**
- **No clear strings in the whole PE.**
- **Few Import (not relevant in .net)**
- **Unusual segment names.**
  - **Home made scripts**
    - **https://github.com/Th4nat0s/Chall_Tools**

Identify that your sample is packed

- **A bunch of clues**
  - **None or very few winnt API calls present in the IAT**

```
$rabin2 -i mymalware.exe
[Imports]
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=kernel32.dll_GetModuleHandleA
ordinal=002 plt=0x00000000 bind=NONE type=FUNC name=kernel32.dll_GetProcAddress
ordinal=003 plt=0x00000000 bind=NONE type=FUNC name=kernel32.dll_ExitProcess
ordinal=004 plt=0x00000000 bind=NONE type=FUNC name=kernel32.dll_LoadLibraryA
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=user32.dll_MessageBoxA
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=advapi32.dll_RegCloseKey
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=oleaut32.dll_SysFreeString
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=gdi32.dll_CreateFontA
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=shell32.dll_ShellExecuteA
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=version.dll_GetFileVersionInfoA
ordinal=001 plt=0x00000000 bind=NONE type=FUNC name=mscoree.dll__CorExeMain

11 Imports
```

Identify that your sample is packed

A bunch of clues

- **High section entropy**
- **Unusual small code segments**
- **Unusual segment names**
  - **Home made scripts**
    - **https://github.com/Th4nat0s/Chall_Tools**

```
$peentro.py badfile.exe
Section         Entropy          Size    MD5                                Remark
.text           4.40891301623    4096    3c25c7a8d445ed1528ba543d6ef35b81
.rdata          2.51973214733    4096    774e8378a9026e53a894eb2043a9cc69
.data           0.599092931135   4096    5c22f870e9c25a2e9331ea30ea55b0ee
.CODE           7.85331928916    86016   dfcbb76bec31c0be1091107edb6ce5d8   Unusal Segment,High Entropy
.rsrc           1.12323628339    4096    adfd501e3b4857ad481c68a07e2425f8
.reloc          0.8026442707     4096    5e07aef133521c73130ec441ed9fa82a
```

Identify the packer

Known tools/packers are easy to identify

- **Unix command *file* works «only» for Upx**
- **Some packers (Upx, Vmprotect) cannot pack .NET PE**
- **Yara rules or the old PEid**
  - **https://github.com/Yara-Rules/rules/blob/master/Packers/packer.yar**
  - **https://www.aldeid.com/wiki/PEiD**
- **RDG packer detector**
  - **http://www.rdgsoft.net (Mute the browser !!!)**
- **DIE (DetectItEasy)**
  - **https://github.com/horsicq/Detect-It-Easy | http://ntinfo.biz/**
- **Exeinfo**
  - **http://exeinfo.atwebpages.com/**

Identifier Tools Usage

- **DIE**

```
$./diec  /home/thanat0s/sample0.exe
PE+(64): compiler: Microsoft Visual C/C++(2008)[-]
PE+(64): linker: Microsoft Linker(9.0)[EXE64,console]

$./diec  /home/thanat0s/sample1.exe
PE: protector: ENIGMA(3.70 build 2015.6.14 20:50:1)[-]
PE: compiler: MinGW(-)[-]
PE: linker: GNU Linker(2.25)[EXE32,admin]

$./diec  /home/thanat0s/sample2.exe
PE: packer: UPX(0.39)[NRV,best]
PE: linker: Polink(2.50*)[EXE32]

$./diec  /home/thanat0s/sample3.exe
PE: protector: Confuser(1.X)[-]
PE: library: .NET(v2.0.50727)[-]
PE: linker: Microsoft Linker(8.0)[EXE32]
```

Identifier Tools Usage

- **File**
  - **file badfile.exe**
- **Yara**
  - **yara (peid|packer).yar badfile.exe**
- **Some homemade** (& dirty) **tools**
  - **peentro.py badfile.exe**

```
$peentro.py badfile.exe
Section         Entropy          Size    MD5                                Remark
.text           4.40891301623    4096    3c25c7a8d445ed1528ba543d6ef35b81
.rdata          2.51973214733    4096    774e8378a9026e53a894eb2043a9cc69
.data           0.599092931135   4096    5c22f870e9c25a2e9331ea30ea55b0ee
.CODE           7.85331928916    86016   dfcbb76bec31c0be1091107edb6ce5d8   Unusal Segment,High Entropy
.rsrc           1.12323628339    4096    adfd501e3b4857ad481c68a07e2425f8
.reloc          0.8026442707     4096    5e07aef133521c73130ec441ed9fa82a
```

SNAPSHOT YOUR VM !!

Packed or not packed ?

Packing triage…. http://upload.trollprod.org/samples.zip

	Packed ?	Why ?
Sample A		
Sample B		
Sample C		
Sample D		
Sample E		
Sample F		
Sample G		
Sample H		
Sample I		
Sample J		

	Packed ?	Why ?
Sample K		
Sample L		
Sample M		
Sample N		
Sample O		
Sample P		
Sample Z		

Password is : infected

Packing triage…….

	Packed ?	Why ?
Sample A	No	but a lot of small B64 strings.
Sample B	Yes,	Diec -> Upx
Sample C	Yes,	Diec -> Confuser
Sample D	Yes	No strings.. Ugly in DnSpy.
Sample E	Yes,	Entropy, dual code segs.
Sample F	Yes,	Entropy
Sample G	Yes,	Entropy, weirds segs.
Sample H	No	strings...but imports...
Sample I	Yes,	Entropy in data
Sample J	Yes,	Huge B64 Strings , Ugly in DnSpy
Sample K	Yes,	Entropy, weirds segs.
Sample L	...don’t know…	weird seg.
Sample M	Yes,	Entropy
Sample N	Yes,	~Entropy, weirds segs.
Sample O	Yes,	Entropy ++
Sample P	it’	Notepad :)
Sample Z	Yes,	Diec -> Enigma

.NET Packer UnPacking

Unpacking .NET samples

- **NEVER open a .NET sample in x86dbg...** (it hurts, badly...)
- **Detect .NET type with «file» or «die»**
- **.NET methods and variables are more than often obfuscated**

```
static <Module>()
{
    <Module>.\u200D\u200D\u200E\u200B\u206E\u206E\u200E\u200E\u200C\u202C\u202E\u200C\u206F\u202B\u
      \u206F\u200B\u206E\u200E\u206C\u206E\u202E();
    <Module>.\u202B\u202C\u206C\u202C\u200C\u202B\u206E\u206A\u200E\u206E\u206C\u202C\u202C\u206D\u
      \u200C\u206A\u202A\u200E\u200D\u206A\u202E();
    <Module>.\u206C\u200F\u202D\u206F\u200D\u202E\u202B\u206A\u206D\u200F\u206F\u200F\u206F\u200B\u
      \u200F\u206F\u200D\u200D\u206A\u202B\u202E();
    for (;;)
    {
        IL_0F:
        uint num = 1821188715u;
        for (;;)
        {
            uint num2;
            switch ((num2 = (num ^ 1292207529u)) % 3u)
            {
            case 0u:
                goto IL_0F;
```

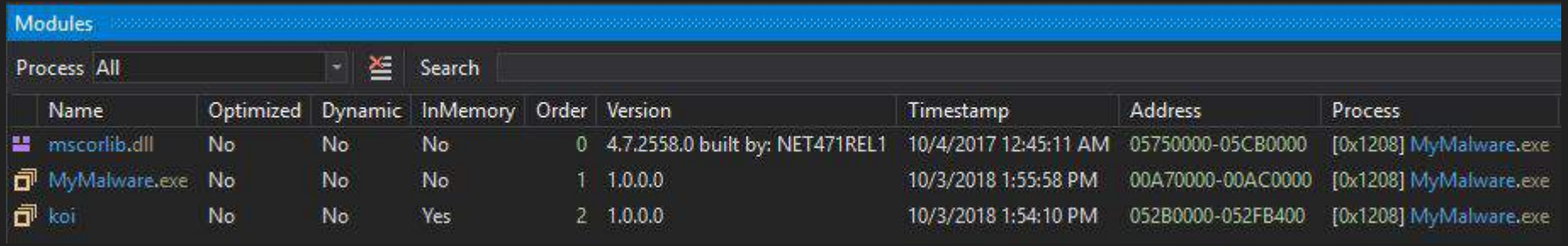
Unpacking .NET samples

Unobfuscation with DE4DOT
https://github.com/0xd4d/de4dot

```
C:\Users\Duke\Desktop>C:\Users\Duke\Documents\RE_Win_Tools\DotNet\De4dot\de4dot.
exe ./mymalware.exe

de4dot v3.1.41592.3405 Copyright (C) 2011-2014 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected .NET Reactor (C:\Users\Duke\Desktop\mymalware.exe)
Cleaning C:\Users\Duke\Desktop\mymalware.exe
Renaming all obfuscated symbols
Saving C:\Users\Duke\Desktop\mymalware-cleaned.exe
```

Unpacking .NET samples

Look for “New modules”



Break and save…

Unpacking .NET samples

Also look for “assembly” or module loading in DnSpy

For us search is “sick”. Use export project / find instead.

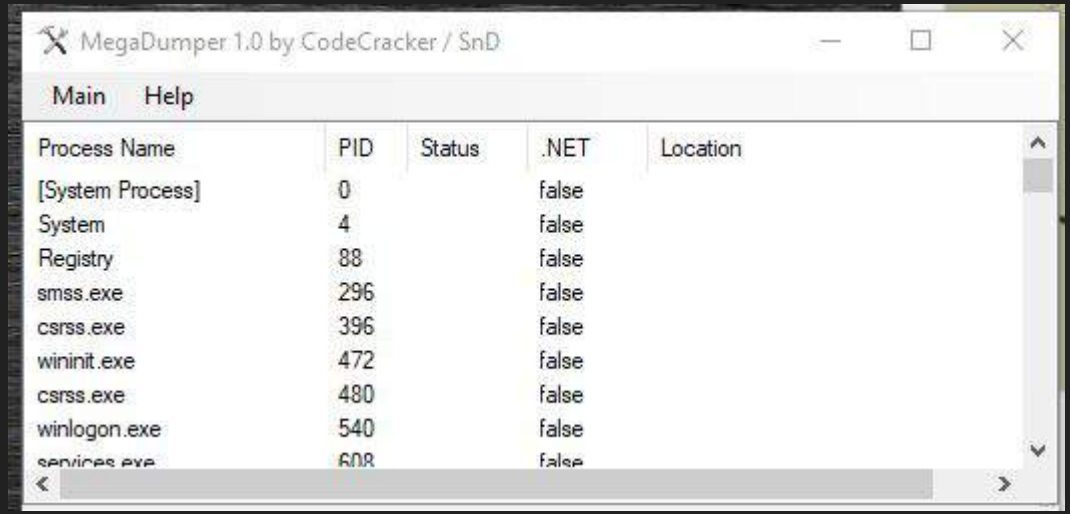
```
// Token: 0x06000005 RID: 5 RVA: 0x0000236C File Offset: 0x0000056C
private static Assembly 乏丢七乒乇(byte[] 主姿乾乞乏)
{
    Type type = Type.GetType("System.Reflection.Assembly");
    MethodInfo method = type.GetMethod("Load", new Type[]
    {
        typeof(byte[])
    });
    return (Assembly)LateBinding.LateGet(method, Type.GetType("System.Reflection.MethodInfo"), "Invoke", new object[]
    {
        null,
        new object[]
        {
            主姿乾乞乏
        }
    }, new string[]
    {
        "obj",
        "parameters"
    }, null);
}
```

Break and save…

Unpacking .NET samples

MegaDumper is a nice tool to dump .NETPE

https://github.com/CodeCracker-Tools/MegaDumper



Run and dump...

When possible,

Fetch sources, not compiled code

Some languages are reversible…

Again, don’t try it in IDA, it hurts… With a good tool, you will retrieve sources

- **Python**
  - **Unpy2exe then uncompyle2 (or Py2ExeBinary Editor)**
- **AutoIT**
  - **exe2aut.exe**
- **AutoHotKey (AHK)**
  - **exe2ahk.exe**

Let’s unpack a .NET !

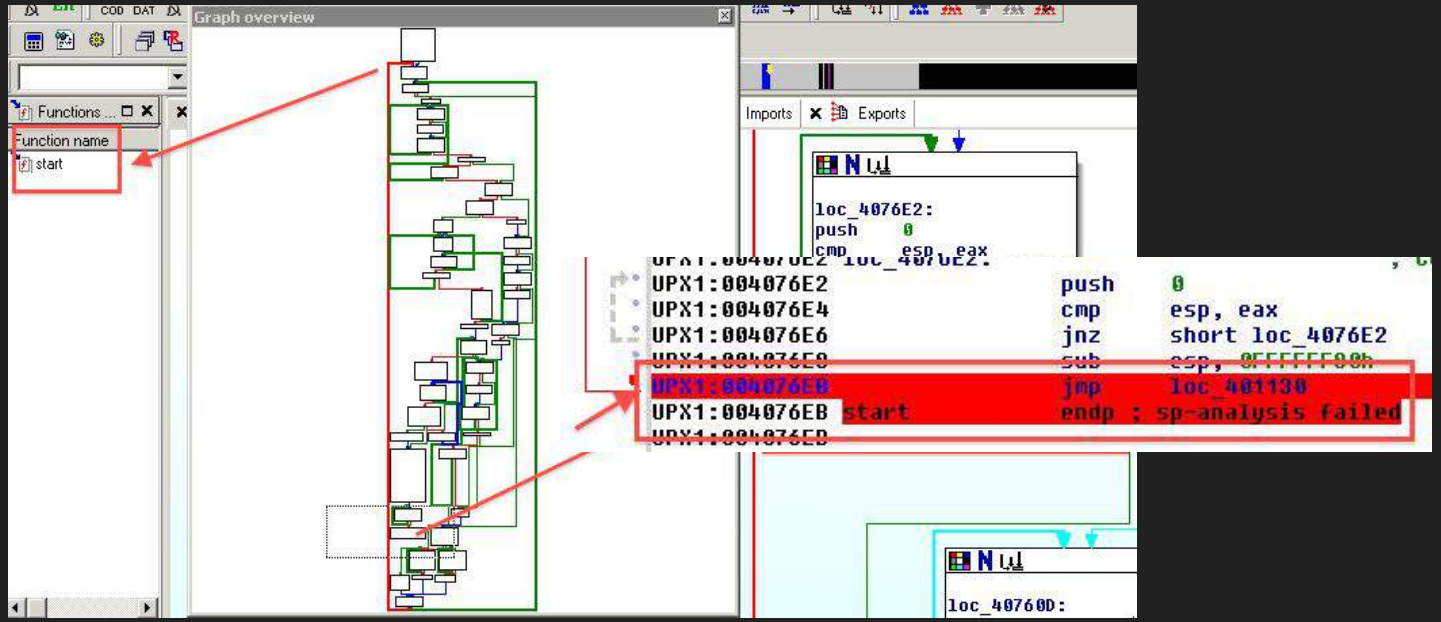
Sample_o.exe
http://upload.trollprod.org/MegaDumper.exe

….. Unpack time

PE Packer UnPacking

“Find the jump” and dump :)

- **Find the jump after unpacking and dump**
- **Prefers hardware breakpoint since the code may move.**



“Find the stack gap” and dump :)

- **Ideal scenario**
  - **Find the pushad/popad after unpacking and dump**
  - **Prefers hardware breakpoint**
  - **Only 32 bits code**

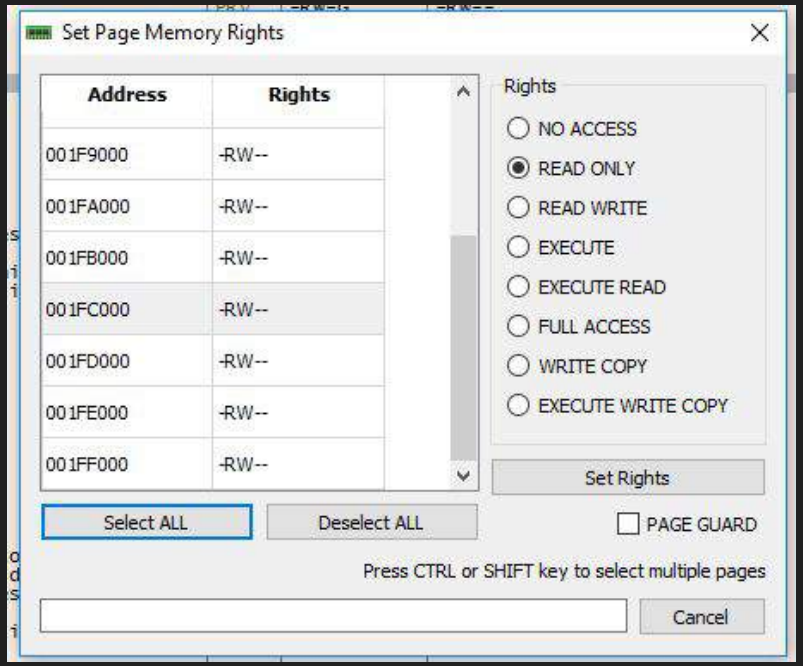
Endless loop trick

- **Find the SetThreadContext call, and note the address of the CONTEXT structure.**
- **Find the child process EntryPoint at CONTEXT + 0xB0, open the suspended process with HxD or ProcessHacker.**
- **Change the opcode by ED FE (jmp eip) and launch the debugged process.**
- **Now you can attach to the child process, replace the jmp by the original opcode.**

- **The pain point is, your VM could run slowly (it’s an endless loop) use multiple CPUs.**

“Find the new RWX segment” and dump :)

- **Break on new RWX segment creation**
  - **Convert it to RW and wait the exception.**



But dumping is not that simple…

Rebuilding

- **IAT**
- **IEP**

Simply “Break” and dump :)

- **Find the unciphered protected PE in a memory segment**
  - **Break on**
    - **WriteProcessMemory**
    - **VirtualAlloc**
    - **VirtualAllocEx**
    - **MapViewOfFile**
    - **UnmapViewOfFile**
    - **….. A lot of them**

Simply “Break” and dump :)

- **Be careful, sometimes the packer use the undocumented API**
    - **Kernel32.WriteProcessMemory**
        - **call ntdll.NtWriteVirtualMemory**
- **Why not calling directly NtWriteVirtualMemory ?**
- **Why not calling the alias ZwWriteVirtualMemory ?**

https://undocumented.ntinternals.net/

Let’s unpack a RunPE !

Sample_n.exe

….. Unpack time

BreakPoint on kernel32!WriteProcessMemory

Going further....

VM Based and Pro packers

Not so easy to extract…
VMProtect http://vmpsoft.com/
TheMida : https://www.oreans.com/themida.php

Real life is sometimes more complicated…
A lot of anti-debugging hidden in the code :)
Look at stack trace, find and bypass them...

Sometimes you may be successful…

Have Fun with samples…

Could you do the unpack challenge ?

WorkShop yourself !!

Easy :

Sample_N
Sample_E
Sample_F
Sample_L
Sample_J

Medium:

Sample_B
Sample_D
Sample_M
Sample_K

Hard:

Sample_G
Sample_L
Sample_Z … for fun...

Unpack Challenge for a free Beer ! :

The first one that finish

It starts with : https://futex.re/ctf/readme.lnk

Droppers if you have time (easy):

SSample_A.doc
SSample_B.doc
SSample_C.vbs
SSample_D.docx
SSample_E.vbe
SSample_F.js
SSample_G.pdf

Contact

Paul Jung
@_ _Thanat0s_ _
pjung@excellium-services.com
www.excellium-services.com

Remi Chipaux
www.qintel.com/company