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](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

File.exe

MZ  DOS Header  PE  PE Header  .text  .data  .idata

_IMAGE_SECTION_HEADER
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;

Sections

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;
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_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
File Mapping

DOS Header
PE Header
.text
.data
.data

EIP

VIRTUAL MEMORY
Import table

Import table list required functions for the PE.

A DLL is a PE
PEB (Process Environment Block)

- Memory structure with the process states
- Location
  - 32 Bits FS[0x30]
  - 64 Bits GS[0x60]
<table>
<thead>
<tr>
<th>Address</th>
<th>Hex dump</th>
<th>Decoded data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1$</td>
<td>0x00</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>$+2$</td>
<td>0x01</td>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>$+4$</td>
<td>0xFFFFFFFF</td>
<td>0xFFFFFFFF</td>
<td></td>
</tr>
<tr>
<td>$+8$</td>
<td>0x00000000</td>
<td>offset nop.&lt;STRUCT IMAGE_DOS_HEADER&gt;</td>
<td></td>
</tr>
<tr>
<td>$+10$</td>
<td>0x0002E400</td>
<td>0x0002E400</td>
<td></td>
</tr>
<tr>
<td>$+14$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+18$</td>
<td>0x00140000</td>
<td>0x00140000</td>
<td></td>
</tr>
<tr>
<td>$+1C$</td>
<td>0x01009C7C</td>
<td>ntdll.RtEnterCriticalSection</td>
<td></td>
</tr>
<tr>
<td>$+20$</td>
<td>0x01009C7C</td>
<td>ntdll.RtLeaveCriticalSection</td>
<td></td>
</tr>
<tr>
<td>$+24$</td>
<td>0x01000001</td>
<td>0x01000001</td>
<td></td>
</tr>
<tr>
<td>$+28$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+2C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+30$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+34$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+38$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+3C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+40$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+44$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+48$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+4C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+50$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+54$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+58$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+5C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+60$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+64$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+68$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+6C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+70$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+74$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+78$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+7C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+80$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+84$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+88$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+8C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+90$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+94$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+98$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+9C$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+A0$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+A4$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+A8$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+AC$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+B0$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+B4$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+B8$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+BC$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+C0$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+C4$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+C8$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+CC$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+D0$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+D4$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+D8$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+DC$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+E0$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+E4$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+E8$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+EC$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+F0$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+F4$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+F8$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>$+FC$</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td></td>
</tr>
</tbody>
</table>

InheritedAddressSpace = 0
ReadImageFileExecutionOptions = 0
BeingDebugged = TRUE
SpareBool = FALSE
Mutant = INVALID_HANDLE_VALUE
ImageBaseAddress = 0x04000000
LoaderData = 0x241E0C
ProcessParameters = 0x00200000
SubsystemData = NULL
ProcessHeap = 0x00140000
FastPebLockRoutine = ntdll.RtEnterCriticalSection
FastPebUnlockRoutine = ntdll.RtLeaveCriticalSection
EnvironmentUpdateCount = 0
KernelCallbackTable = NULL
Reserved = 0
ThunkOrOptions = 0
FreeRaw = 0
TlsExpansionCounter = 0
TlsBitmap = ntdll.7C9805E0
TlsBitmapBits[2] = 1
ReadOnlySharedMemoryBase = 7F6F0000
ReadOnlySharedMemoryHeap = 7F6F0000
ReadOnlyStaticServerData = 7F6F0000
AmsiCodePageData = 7FFB0000
OemCodePageData = 7FFC1000
UnicodeCaseTableData = 7FFD2000
NumberOfProcessors = 2
NtGlobalFlag = 112.
Reserved = 0
CriticalSectionTimeout.Lo = 7F800000
CriticalSectionTimeout.Hi = -1799
HeapSegmentReserve = 184576E.
HeapSegmentCommit = 0192.
HeapDeCommitTotalFreeThreshold = 65536.
HeapDeCommitFreeBlockThreshold = 498.
NumberOfHeaps = 3
MaxNumNumberOfHeaps = 16.
ProcessHeaps = 7C97FFE0
GdiSharedHandleTable = NULL
ProcessStarterAddress = NULL
GdiDCAttributeList = 0
LoaderLock = 7C97E174
OSMajorVersion = 5
OSMinorVersion = 0
OSBuildNumber = 2600.
OSSubsystemVersion = 768.
OSPlatformId = 2
ImageSubsystem = 2
ImageSubsystemMajorVersion = 4
ImageSubsystemMinorVersion = 0
ImageProcessAffinityMask = 0
GdiHandleBuffer[34, ] = 0
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 address of 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

CreateProcess, CREATE_SUSPENDED
GetThreadContext : EBX -> PEB
RunPE

Packer A

Malware B

Executable B

CreateProcess, CREATE_SUSPENDED

GetThreadContext : EBX -> PEB

NtUnmapViewOfSection
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

Packer A

Malware B

VirtualAllocEx
WriteProcessMemory
ResumeThread

Executable B
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 resource

- 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

<table>
<thead>
<tr>
<th>Section</th>
<th>Entropy</th>
<th>Size</th>
<th>MD5</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>4.40891301623</td>
<td>4096</td>
<td>3c25c7a8d445ed1528ba543d6ef35b81</td>
<td></td>
</tr>
<tr>
<td>.rdata</td>
<td>2.51973214733</td>
<td>4096</td>
<td>774e8378a9026e53a894eb2043a9cc69</td>
<td></td>
</tr>
<tr>
<td>.data</td>
<td>0.599092931135</td>
<td>4096</td>
<td>5c22f870e9c25a2e9331ea30ea55b0ee</td>
<td></td>
</tr>
<tr>
<td>.CODE</td>
<td>7.85331928916</td>
<td>86016</td>
<td>dfcbb76bec31c0be1091107edb6ce5d8</td>
<td>Unusual Segment,High Entropy</td>
</tr>
<tr>
<td>.rsrc</td>
<td>1.12323628339</td>
<td>4096</td>
<td>adfd501e3b4857ad481c68a07e2425f8</td>
<td></td>
</tr>
<tr>
<td>.reloc</td>
<td>0.8026442707</td>
<td>4096</td>
<td>5e07aef133521c73130ec441ed9fa82a</td>
<td></td>
</tr>
</tbody>
</table>
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://www.aldeid.com/wiki/PEiD](https://www.aldeid.com/wiki/PEiD)
- RDG packer detector
  - [http://www.rdgsoft.net](http://www.rdgsoft.net) (Mute the browser !!!)
- DIE (DetectItEasy)
- Exeinfo
  - [http://exeinfo.atwebpages.com/](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
```

<table>
<thead>
<tr>
<th>Section</th>
<th>Entropy</th>
<th>Size</th>
<th>MD5</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>4.40891301623</td>
<td>4096</td>
<td>3c25c7a8d445ed1528ba543d6ef35b81</td>
<td></td>
</tr>
<tr>
<td>.rdata</td>
<td>2.51973214733</td>
<td>4096</td>
<td>774e8378a9026e53a894eb2043a9cc69</td>
<td></td>
</tr>
<tr>
<td>.data</td>
<td>0.599092931135</td>
<td>4096</td>
<td>5c22f870e9c25a2e9331ea30ea55b0ee</td>
<td></td>
</tr>
<tr>
<td>.CODE</td>
<td>7.85331928916</td>
<td>86016</td>
<td>dfcbb76bec31c0be1091107edb6ce5d8</td>
<td>Unusual Segment,High Entropy</td>
</tr>
<tr>
<td>.rsrc</td>
<td>1.12323628339</td>
<td>4096</td>
<td>adfd501e3b4857ad481c68a07e2425f8</td>
<td></td>
</tr>
<tr>
<td>.reloc</td>
<td>0.8026442707</td>
<td>4096</td>
<td>5e07aef133521c73130ec441ed9fa82a</td>
<td></td>
</tr>
</tbody>
</table>
SNAPSHOT YOUR VM !!
Packed or not packed ?
Packing triage.... http://upload.trollprod.org/samples.zip

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample A</td>
<td></td>
<td>Sample K</td>
<td></td>
</tr>
<tr>
<td>Sample B</td>
<td></td>
<td>Sample L</td>
<td></td>
</tr>
<tr>
<td>Sample C</td>
<td></td>
<td>Sample M</td>
<td></td>
</tr>
<tr>
<td>Sample D</td>
<td></td>
<td>Sample N</td>
<td></td>
</tr>
<tr>
<td>Sample E</td>
<td></td>
<td>Sample O</td>
<td></td>
</tr>
<tr>
<td>Sample F</td>
<td></td>
<td>Sample P</td>
<td></td>
</tr>
<tr>
<td>Sample G</td>
<td></td>
<td>Sample Z</td>
<td></td>
</tr>
<tr>
<td>Sample H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sample I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sample J</td>
<td></td>
<td></td>
<td>Password is: infected</td>
</tr>
</tbody>
</table>
Packing triage

<table>
<thead>
<tr>
<th>Sample</th>
<th>Packed?</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample A</td>
<td>No</td>
<td>but a lot of small B64 strings.</td>
</tr>
<tr>
<td>Sample B</td>
<td>Yes</td>
<td>Diec -&gt; Upx</td>
</tr>
<tr>
<td>Sample C</td>
<td>Yes</td>
<td>Diec -&gt; Confuser</td>
</tr>
<tr>
<td>Sample D</td>
<td>Yes</td>
<td>No strings.. Ugly in DnSpy.</td>
</tr>
<tr>
<td>Sample E</td>
<td>Yes</td>
<td>Entropy, dual code segs.</td>
</tr>
<tr>
<td>Sample F</td>
<td>Yes</td>
<td>Entropy</td>
</tr>
<tr>
<td>Sample G</td>
<td>Yes</td>
<td>Entropy, weirds segs.</td>
</tr>
<tr>
<td>Sample H</td>
<td>No</td>
<td>strings...but imports...</td>
</tr>
<tr>
<td>Sample I</td>
<td>Yes</td>
<td>Entropy in data</td>
</tr>
<tr>
<td>Sample J</td>
<td>Yes</td>
<td>Huge B64 Strings , Ugly in DnSpy</td>
</tr>
</tbody>
</table>

| 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
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.
Unpacking .NET samples

MegaDumper is a nice tool to dump .NET PE

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