HACK.LU

Unpacking Cheat Sheet v1.0

Get strings

```
strings mysample
strings -el mysample (for unicode)
```

Packer Identification

```
file mysample
diec mysample
peentro.py mysample
yara (peid|packer).yar mysample
```

Retrieve section names

```
rabin2 -S mysample
peentro.py mysample
```

Usual sections name

.bss	.code/.text	.data
.debug	.edata	.idata
.pdata	.rdata	.reloc
.rsrc	.tls	.xdata

Extract PE from a dump

```
decrottePE.py mydump
```

Retrieve IAT table

```
rabin2 -i mysample
```

Direct API call

```
Fs[0x30] 32 Bits PEB location
Gs[0x60] 64 Bits PEB location
kernel32!GetProcAddress(A|W)
```

Classical Find DLLs

```
Get PEB, then LDR then InMemoryOrder
mov reg32, [fs:0x30]     get PEB
mov reg32, [reg32+0x0C]  get LDR
mov reg32, [reg32+0x14]  get InMe…
```

Process Hollowing APIs

Create a process

```
kernel32!CreateProcess(A|W)
ntdll!(Nt|Zw)CreateProcess
ntdll!(Nt|Zw)CreateProcessEx
```

Read/write registers in process

```
kernel32!GetThreadContext (EBX⇨PEB)
kernel32!SetThreadContext (EAX⇦EIP)
ntdll!(Nt|Zw)GetContextThread
```

Write/read/set memory

```
kernel32!CreateFileMapping(A|W)
ntdll!(Nt|Zw)UnmapViewOfSection
ntdll!(Nt|Zw)UnmapViewOfSectionEx
kernel32!VirtualAllocEX
kernel32!(Write|Read)ProcessMemory
ntdll!(Nt|Zw)WriteVirtualMemory
ntdll!(Nt|Zw)ProtectVirtualMemory
```

Resume a process

```
kernel32!ResumeThread
ntdll!(Nt|Zw)ResumeThread
```

Suspend a process

```
ntdll!(Nt|Zw)SuspendProcess
ntdll!(Nt|Zw)SuspendThread
kernel32!SuspendThread
kernel32!WoW64SuspendThread
```

Mem. Prot. Constants

```
0x01 PAGE_NOACCESS
0x02 PAGE_READONLY
0x04 PAGE_READWRITE
0x08 PAGE_WRITECOPY
0x10 PAGE_EXECUTE
0x20 PAGE_EXECUTE_READ
0x40 PAGE_EXECUTE_READWRITE
0x80 PAGE_EXECUTE_WRITECOPY
```

x86dbg Basics

F2	Toggle breakpoint
Shift+F2	Set conditional breakpoint
Ctrl+F2	Restart
F9	Run
Shift+F9	Run and pass exception
Ctrl+F9	Execute till return
F7	Step into
F8	Step over
F4	Run to selection
Ctrl+A	Analyse code
Ctrl+C	Copy selection
Ctrl+E	Edit in binary format
Ctrl+F	Search for a command
Ctrl+G	Goto expression
*	Goto Origin
Ctrl+I	Launch dumper (Scyla)
Ctrl+K	Go to previous reference
ALT+G	Toggle graph view
ALT+M	Toggle Memory Map view
ALT+E	Toggle Modules API
Enter	Follow jump or call
:	Add label

;	Add comment

DnSpy Basics

F9	Toggle breakpoint
F5	Run
F10	Step over
F11	Step into
Shit-F11	Step out
Modules	Ctrl+Alt+U
Memory 1	Alt+6
Memory x	Ctrl+Shift+x
Process	Ctrl+Alt+z

ASM Basics

Register size

x32 ABCD (E**A**X, E**B**X…, **A**X, **B**X…)
x64 ABCD 8-15 (R**A**X, R**8**…, E**A**X, R**8**D…)

	Letter	Num
0x1234567812345678	R**.**X	R**.**
0x12345678	E**.**X	R**.**D
0x5678	**.**X	R**.**W
0x56	**.**H	
0x78	**.**L	R**.**L

Registers functions

X64

RAX return value
RBX general purpose
RCX counter / arg1
RDX math / arg2
R8 arg3
R9 arg4
R10-15 general purpose
RSP Stack pointer
RBP Base pointer
Other arguments on stack

X86

EAX return value
EBX general purpose
ECX counter
EDX math, EAX couple
ESP Stack pointer
EBP Base pointer
Arguments all on stack (stdcall)
Arguments ECX, EDX,stack (fastcall)

Opcodes

Opc dst, src : Asm Intel syntax
MOV : Move (copy)
REP : Repeat [ER]CX related
MOVS[BWQ] : Move memory array
LODS[BWQ] : Load memory array
STOS[BWQ] : Write memory array
XCHG : Exchange
CMP : Compare values
PUSH : Push onto stack
POP : Fetch from stack
ADD : Add
SUB : Subtract
DIV : Divide
IDIV : Signed integer divide
MUL : Multiply
IMUL : Signed integer multiply
INC : Increment
DEC : Decrement
SAL : Shift left
SAR : Shift right
SHL : Shift logical left
SHR : Shift logical right
ROL : Rotate left
ROR : Rotate right
NOT : Invert each bit
AND : Logical and
OR   : Logical or
XOR : Logical exclusive or
NOP : No operation (0x90)
INT : Interrupt
CALL : Call subroutine
RET : Return to function
LOOP : Loop relate with [ER]cx
SYSENTER: Kernel fastcall
JMP : goto

Conditionals jumps:

Syntax is : *J(N)X(X)*
N is "not"
Where X is :

E equal	**Z** Zero	**A** Above
B Below	**C** Carry	**G** Greater
L lower	**O** Overflow	**S** Signed

Example:
JNEA : Jump if not equal or above
JB: Jump if below
JGE: Jump if greater or equal

Extra :
JCXZ : Jump if CX zero
JECXZ: Jump if ECX zero
JRCXZ: Jump if RCX zero