MAKE DKOM ATTACKS GREAT AGAIN

MARIANO GRAZIANO

HackInBo Winter 2016 Edition

Bologna, Italy - 29/10/2016

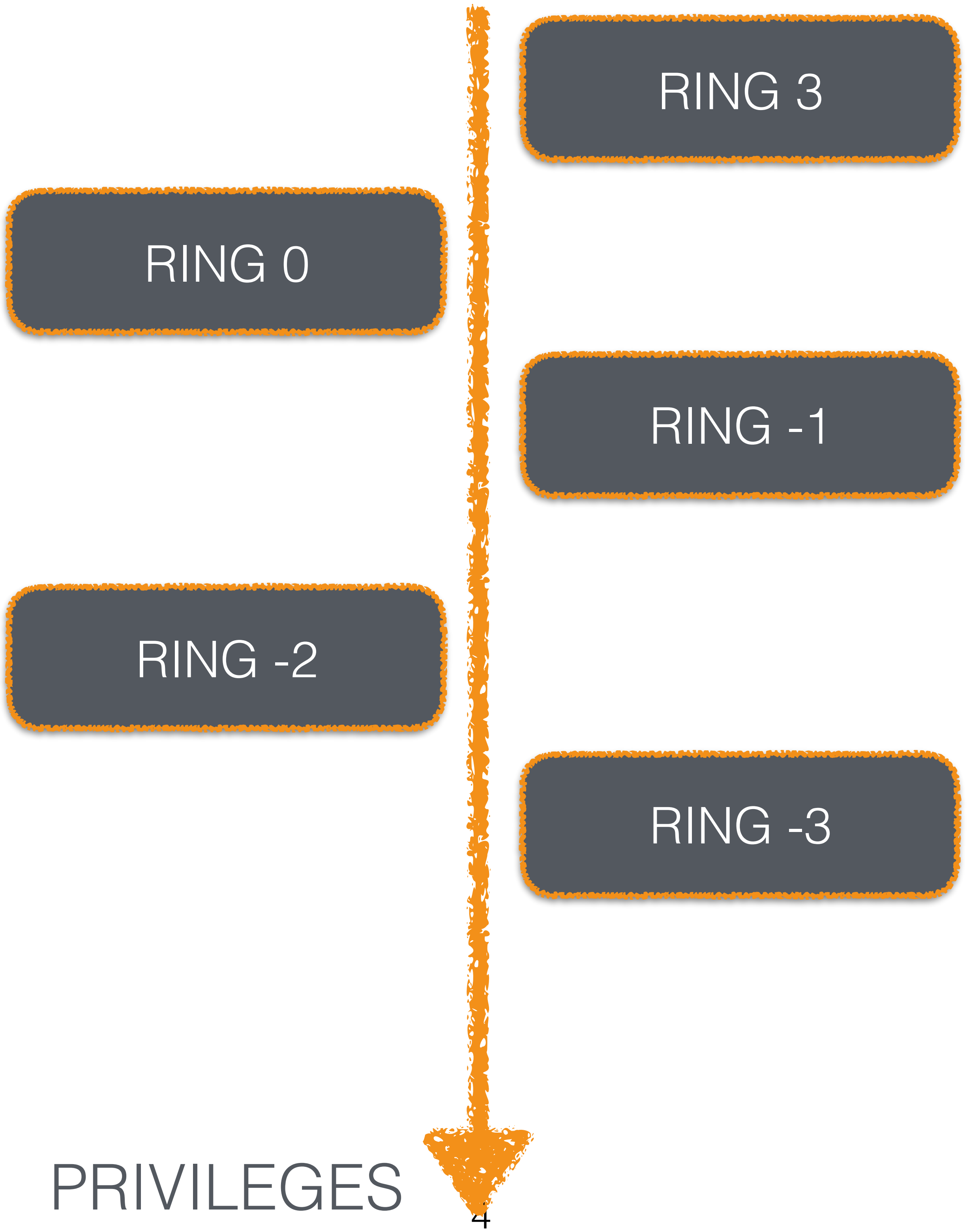
whoami

- Security researcher at Cisco in the Talos group
- Ph.D. Telecom ParisTech/Eurecom
- Hackademic
- Malware analysis / memory forensics

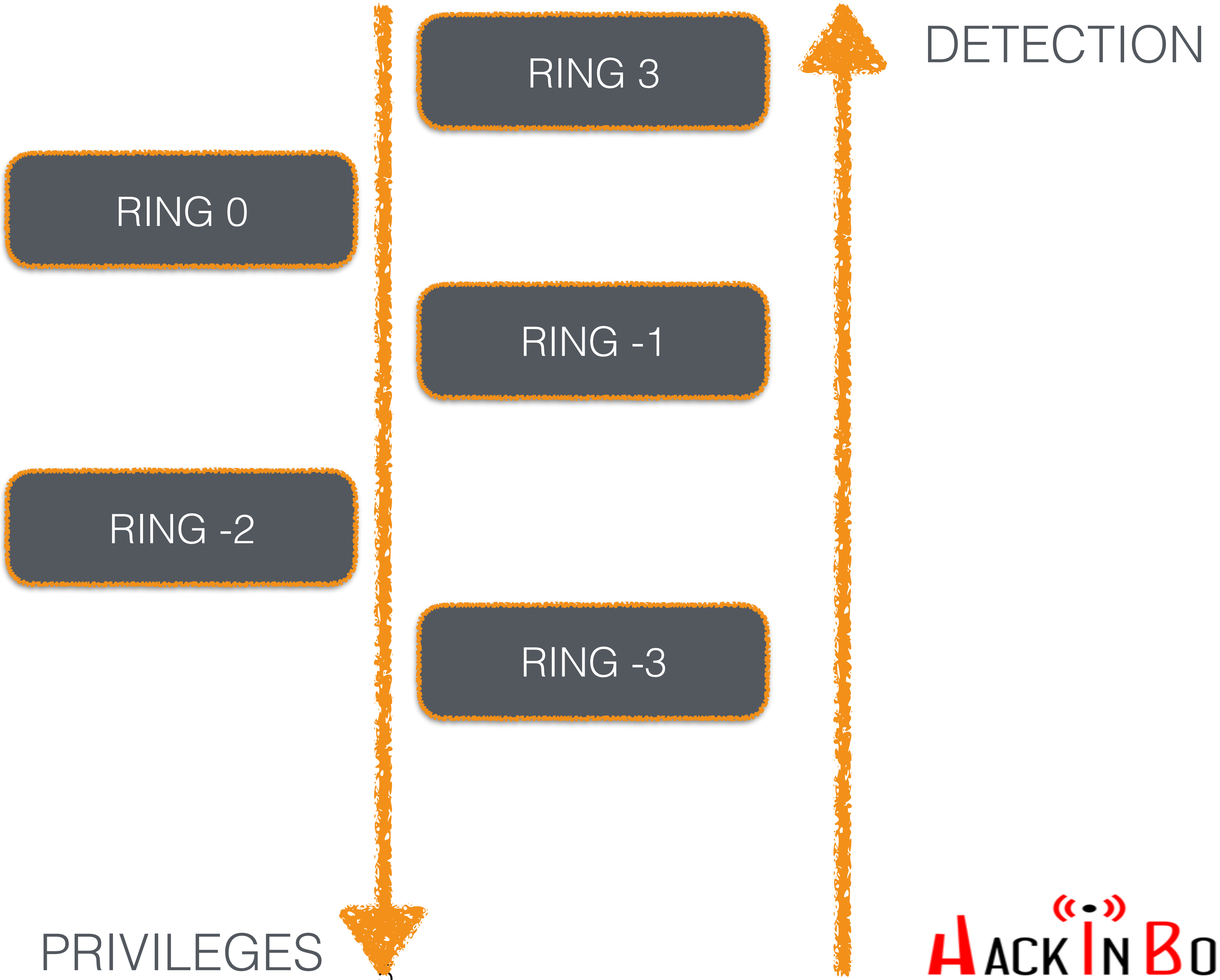
ROOTKIT

"Software to maintain a **persistent** and **stealthy** access on a compromised machine"

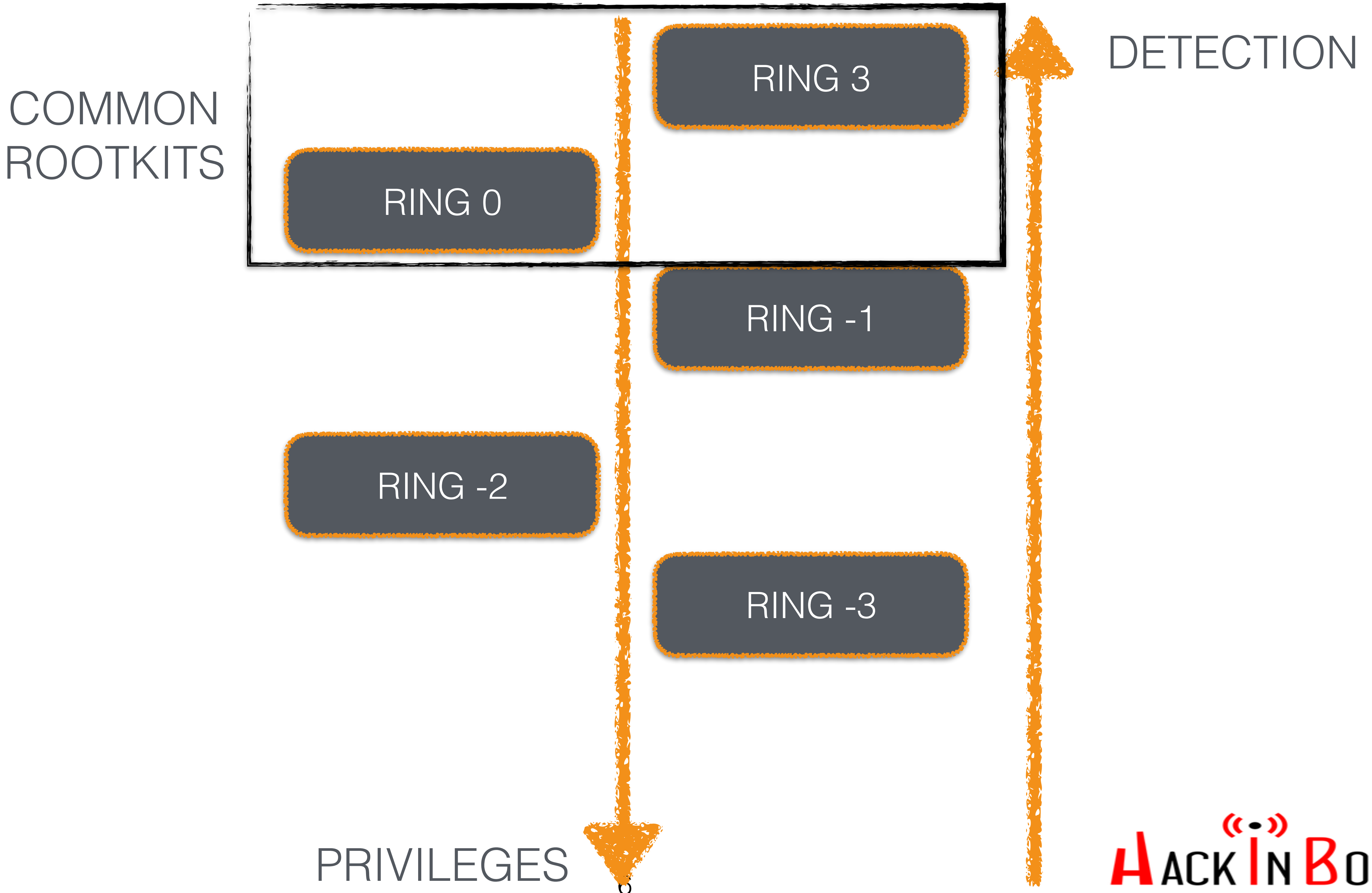
HOW?



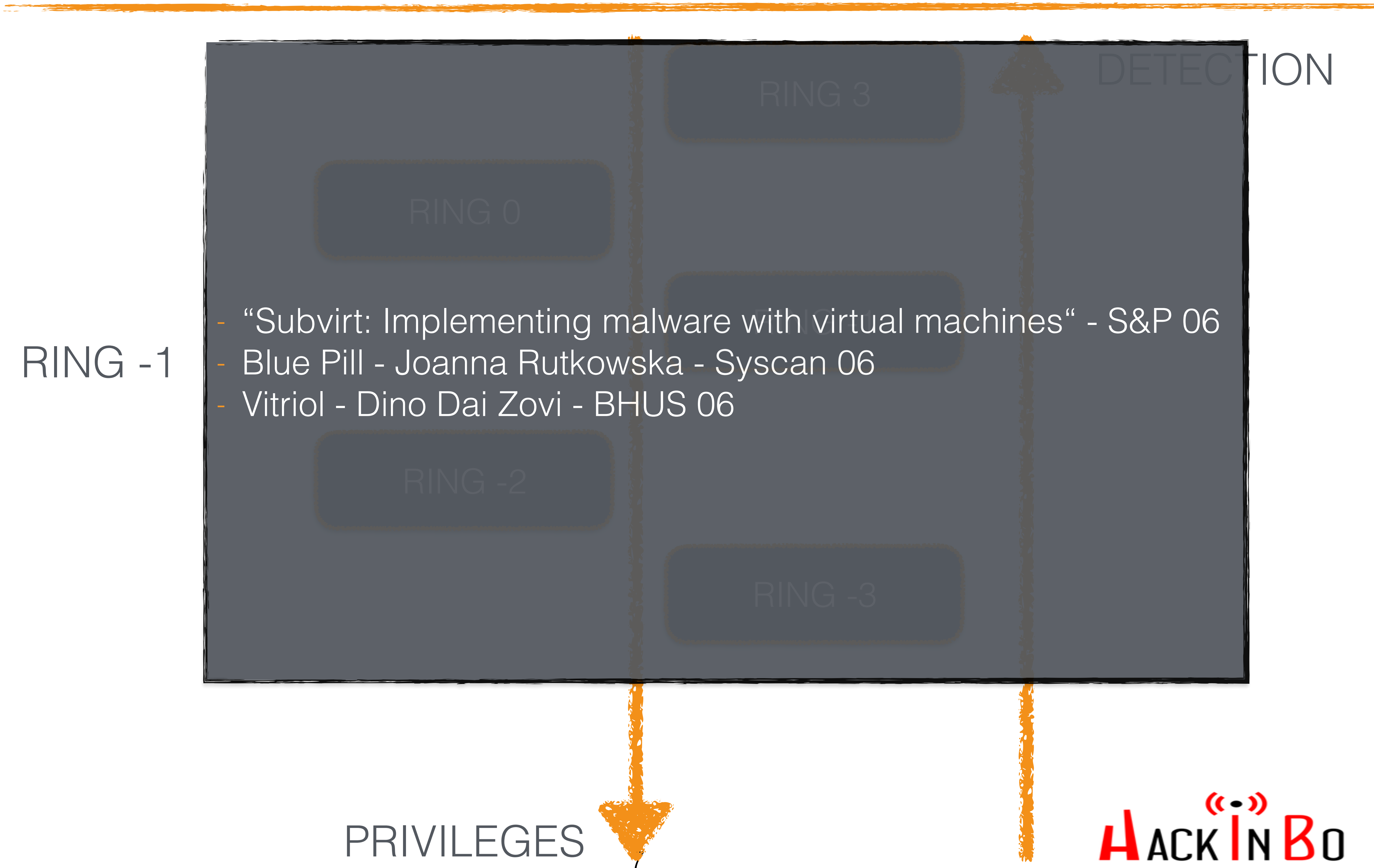
HOW?



HOW?


COMMON ROOTKITS
RING 3
RING 0
RING -1
RING -2
RING -3
DETECTION
PRIVILEGES

HOW?



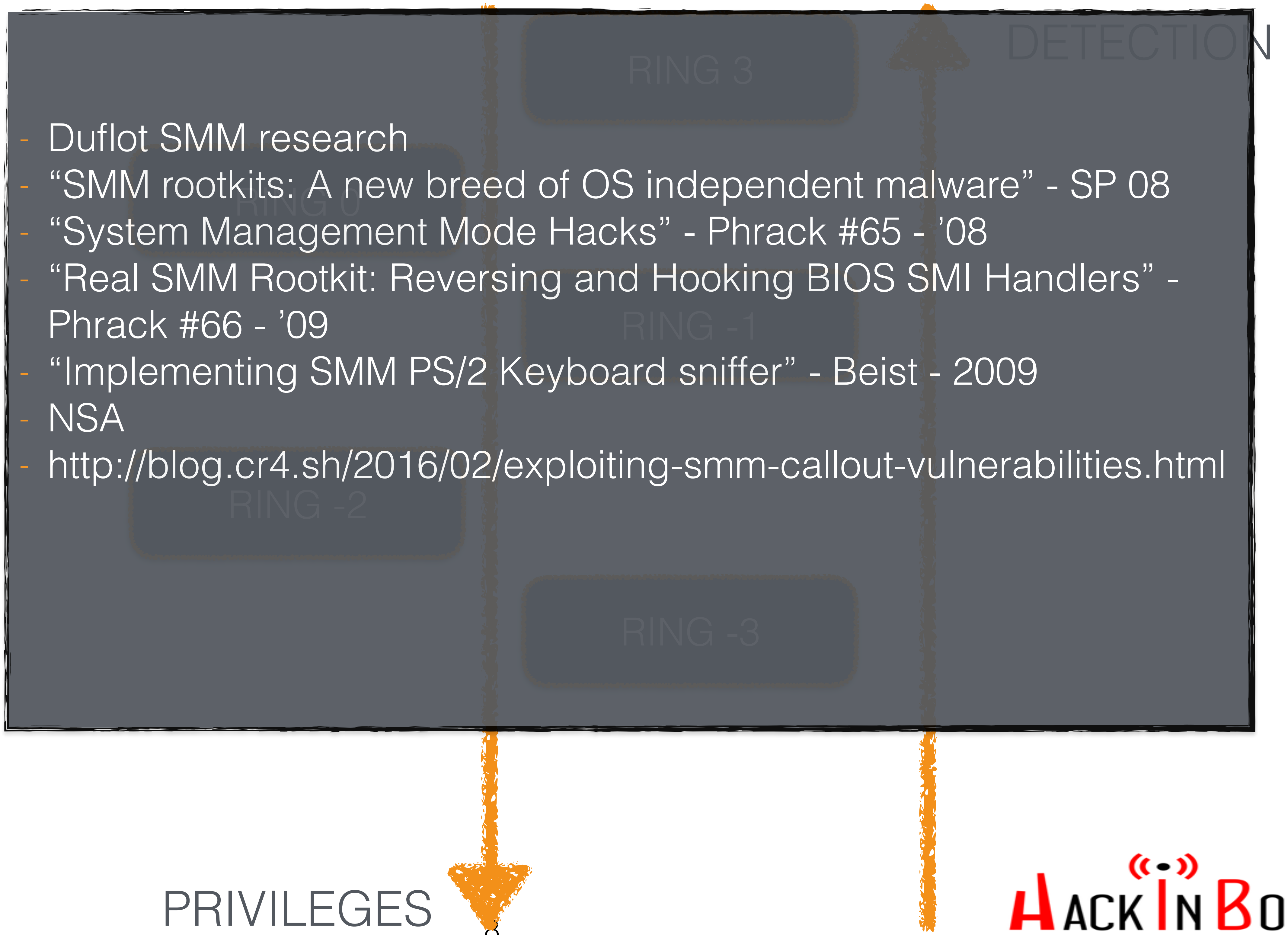
RING -1

- "Subvirt: Implementing malware with virtual machines" - S&P 06
- Blue Pill - Joanna Rutkowska - Syscan 06
- Vitriol - Dino Dai Zovi - BHUS 06

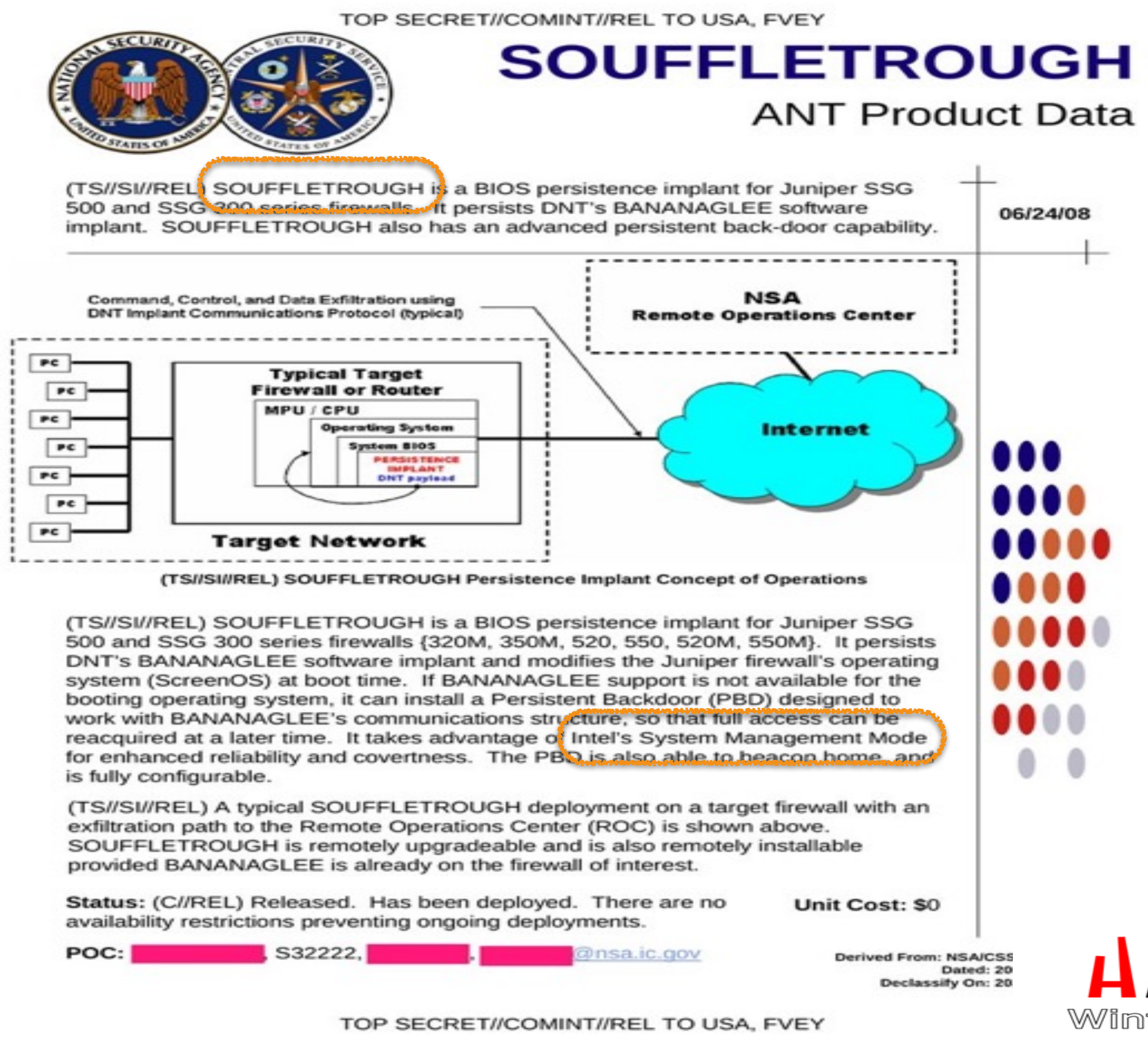
HOW?

RING -2

- Duflot SMM research
- “SMM rootkits: A new breed of OS independent malware” - SP 08
- “System Management Mode Hacks” - Phrack #65 - ’08
- “Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers” - Phrack #66 - ’09
- “Implementing SMM PS/2 Keyboard sniffer” - Beist - 2009
- NSA
- http://blog.cr4.sh/2016/02/exploiting-smm-callout-vulnerabilities.html



HOW?



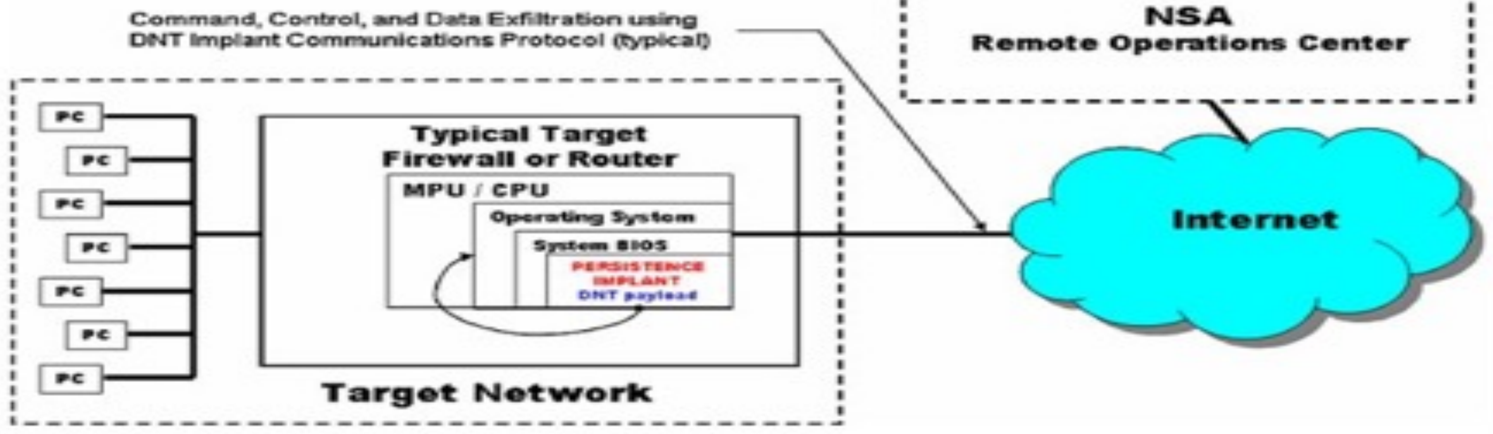
TOP SECRET//COMINT//REL TO USA, FVEY

SOUFFLETROUGH

ANT Product Data

(TS//SI//REL) SOUFFLETROUGH is a BIOS persistence implant for Juniper SSG 500 and SSG 300 series firewalls. It persists DNT's BANANAGLEE software implant. SOUFFLETROUGH also has an advanced persistent back-door capability.

06/24/08


Command, Control, and Data Exfiltration using DNT Implant Communications Protocol (typical)
NSA Remote Operations Center
Internet
Typical Target Firewall or Router
MPU / CPU
Operating System
System BIOS
PERSISTENCE IMPLANT
DNT payload
PC
PC
PC
PC
PC
PC
PC
Target Network

(TS//SI//REL) SOUFFLETROUGH Persistence Implant Concept of Operations

(TS//SI//REL) SOUFFLETROUGH is a BIOS persistence implant for Juniper SSG 500 and SSG 300 series firewalls {320M, 350M, 520, 550, 520M, 550M}. It persists DNT's BANANAGLEE software implant and modifies the Juniper firewall's operating system (ScreenOS) at boot time. If BANANAGLEE support is not available for the booting operating system, it can install a Persistent Backdoor (PBD) designed to work with BANANAGLEE's communications structure, so that full access can be reacquired at a later time. It takes advantage of Intel's System Management Mode for enhanced reliability and covertness. The PBD is also able to beacon home, and is fully configurable.

(TS//SI//REL) A typical SOUFFLETROUGH deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. SOUFFLETROUGH is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

Status: (C//REL) Released. Has been deployed. There are no availability restrictions preventing ongoing deployments.

Unit Cost: $0

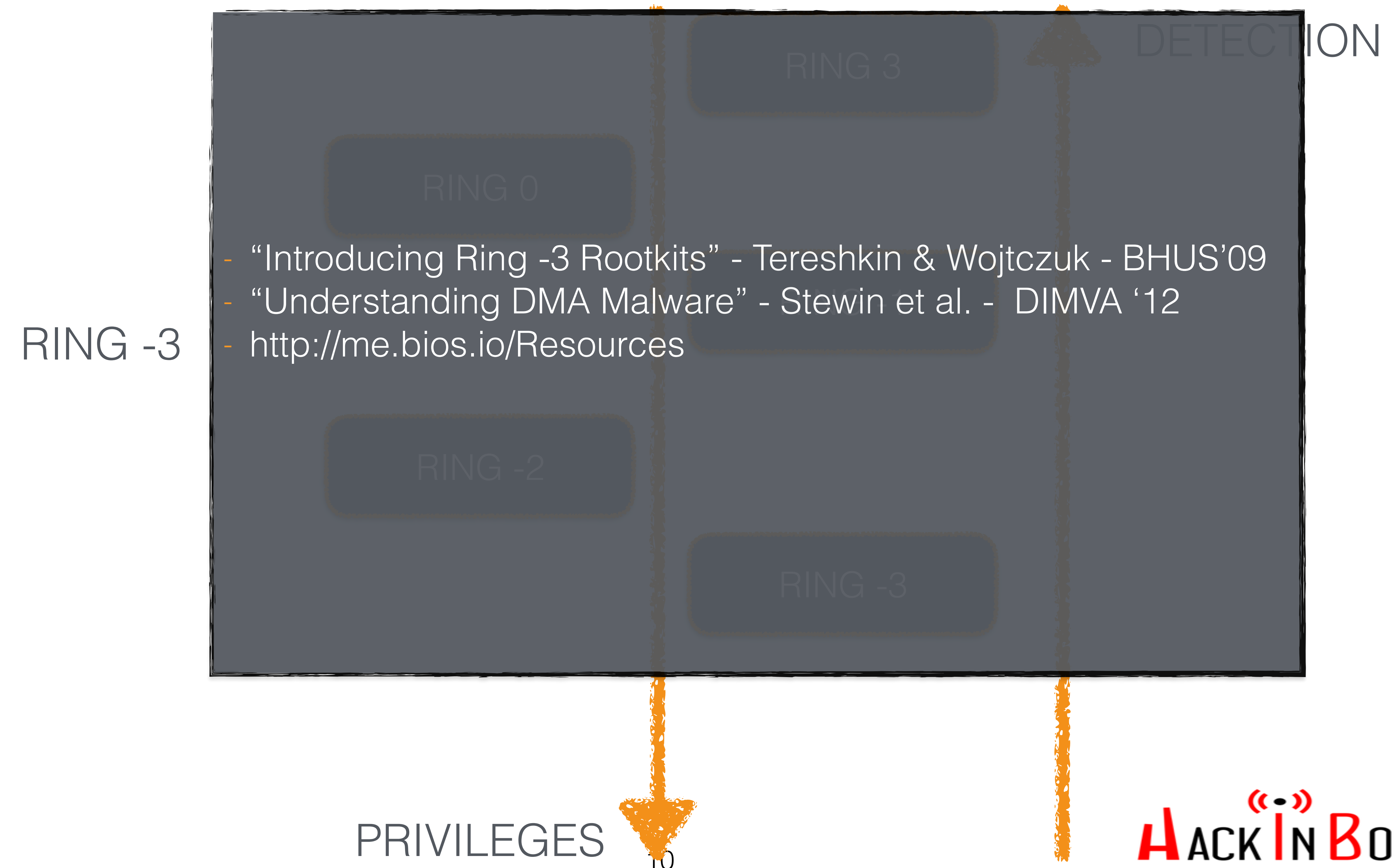
POC: [redacted], S32222, [redacted], [redacted]@nsa.ic.gov

Derived From: NSA/CSS
Dated: 20
Declassify On: 20

TOP SECRET//COMINT//REL TO USA, FVEY

HACK IN BO Winter 2016 Edition

HOW?



- "Introducing Ring -3 Rootkits" - Tereshkin & Wojtczuk - BHUS'09
- "Understanding DMA Malware" - Stewin et al. - DIMVA '12
- http://me.bios.io/Resources

HOW?

DKOM

BOOTKITS

ROP ROOTKITS

BLUEPILLS

FIRMWARE

HOW?

DKOM

BOOTKITS

ROP ROOTKITS

BLUEPILLS

FIRMWARE

ROP ROOTKIT?

- Motivation
- “*Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms*” - USENIX Security 09
- “*Persistent Data-only Malware: Function Hooks without Code*” - NDSS ‘14

ROP ROOTKIT?

- Persistence technique:
  - CVE-2013-2094
  - `sysenter`
    - `IA32_SYSENTER_ESP (0x175)`
    - `IA32_SYSENTER_EIP (0x176)`

ROP ROOTKIT?

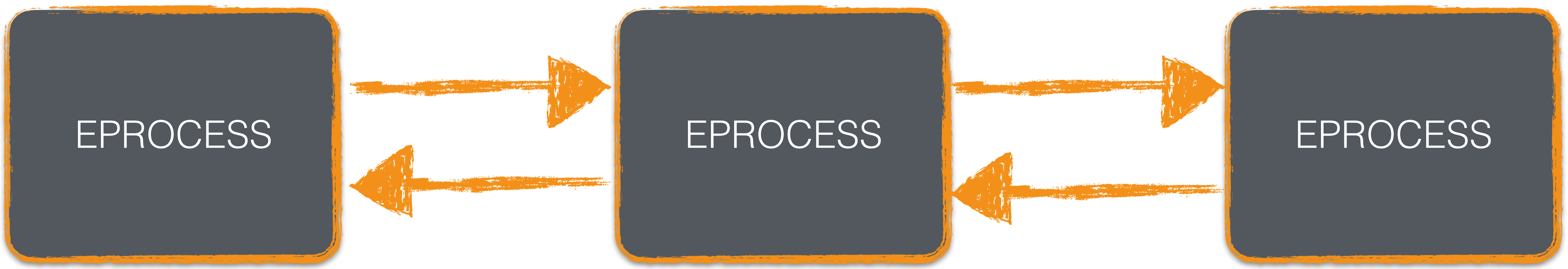
Chuck ROP chains:

CHAIN	INSTRUCTIONS	GADGETS	BLOCKS	BRANCHES	FUNCTIONS	CALLS
COPY	414,275	184,126	1	-	-	-
DISPATCHER	63,515	28,874	7	3	1	5
PAYLOAD	6320	2913	34	26	9	17

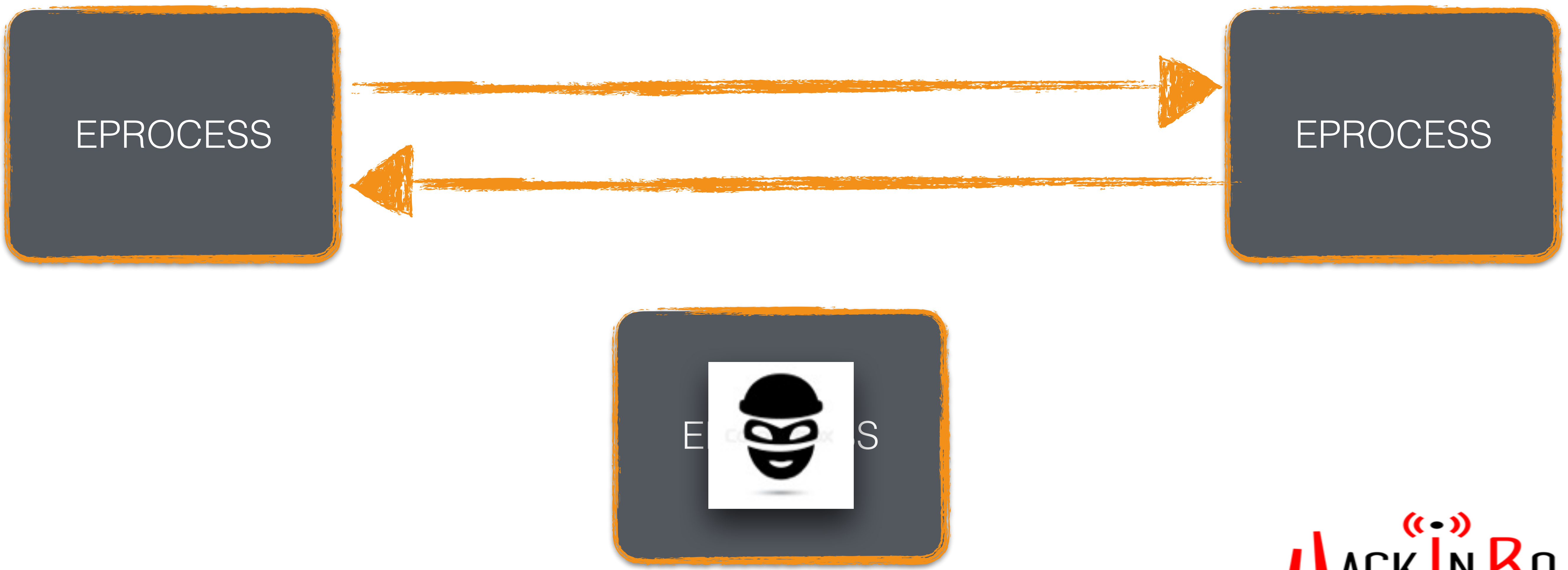
DKOM

“**D**irect **K**ernel **O**bject **M**anipulation”

TRADITIONAL DKOM



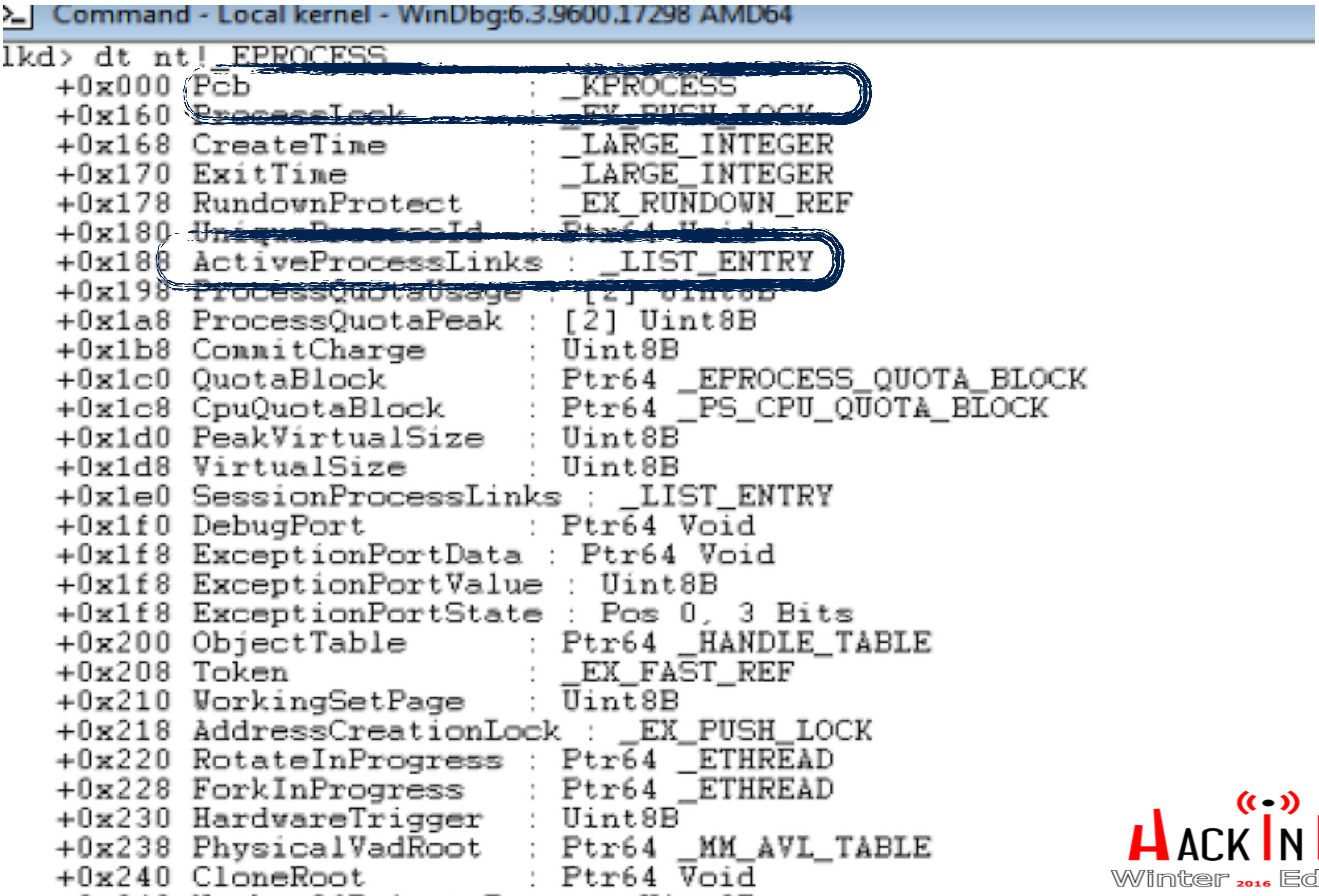
TRADITIONAL DKOM



DKOM vs PROCESSES

- DKOM is a generic technique
- Processes:
  - Windows: `KPROCESS/EPROCESS/PEB`
  - Linux: `task_struct`
  - OSX: `proc/task`

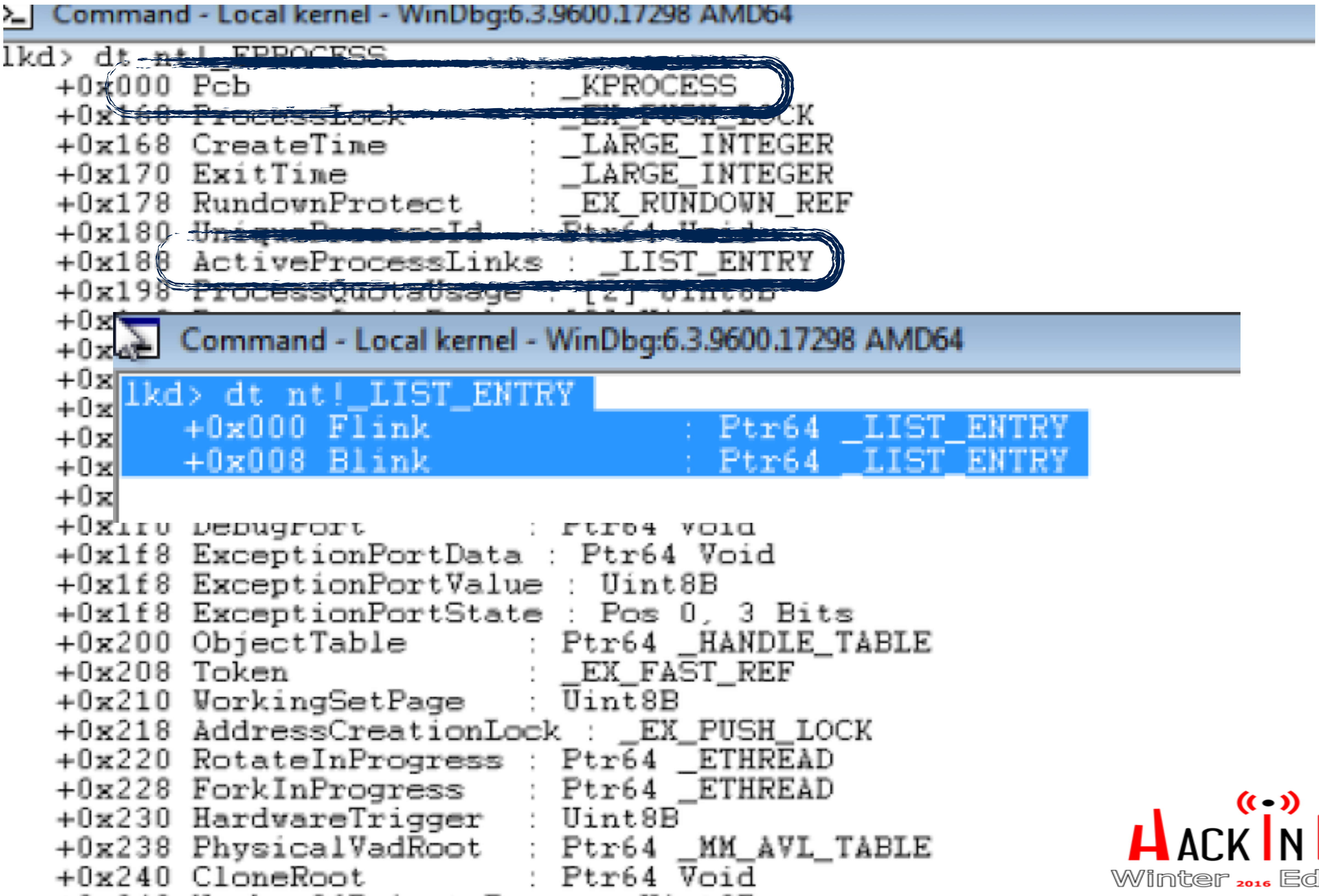
(E)PROCESS?



Command - Local kernel - WinDbg:6.3.9600.17298 AMD64

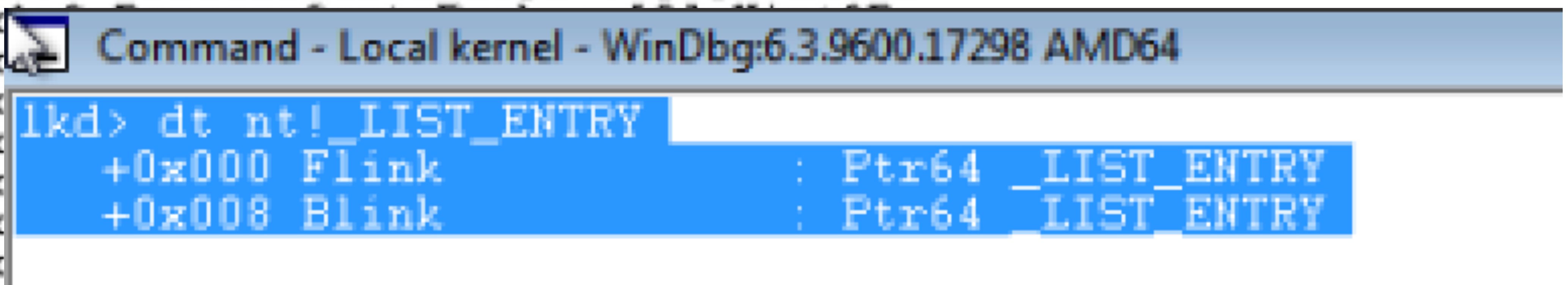
```
lkd> dt nt!_EPROCESS
   +0x000 Pcb                     : _KPROCESS
   +0x160 ProcessLock             : _EX_PUSH_LOCK
   +0x168 CreateTime              : _LARGE_INTEGER
   +0x170 ExitTime                : _LARGE_INTEGER
   +0x178 RundownProtect          : _EX_RUNDOWN_REF
   +0x180 UniqueProcessId         : Ptr64 Void
   +0x188 ActiveProcessLinks : _LIST_ENTRY
   +0x198 ProcessQuotaUsage : [2] Uint8B
   +0x1a8 ProcessQuotaPeak  : [2] Uint8B
   +0x1b8 CommitCharge            : Uint8B
   +0x1c0 QuotaBlock              : Ptr64 _EPROCESS_QUOTA_BLOCK
   +0x1c8 CpuQuotaBlock           : Ptr64 _PS_CPU_QUOTA_BLOCK
   +0x1d0 PeakVirtualSize         : Uint8B
   +0x1d8 VirtualSize             : Uint8B
   +0x1e0 SessionProcessLinks : _LIST_ENTRY
   +0x1f0 DebugPort               : Ptr64 Void
   +0x1f8 ExceptionPortData : Ptr64 Void
   +0x1f8 ExceptionPortValue : Uint8B
   +0x1f8 ExceptionPortState : Pos 0, 3 Bits
   +0x200 ObjectTable             : Ptr64 _HANDLE_TABLE
   +0x208 Token                   : _EX_FAST_REF
   +0x210 WorkingSetPage          : Uint8B
   +0x218 AddressCreationLock : _EX_PUSH_LOCK
   +0x220 RotateInProgress  : Ptr64 _ETHREAD
   +0x228 ForkInProgress          : Ptr64 _ETHREAD
   +0x230 HardwareTrigger         : Uint8B
   +0x238 PhysicalVadRoot         : Ptr64 _MM_AVL_TABLE
   +0x240 CloneRoot               : Ptr64 Void
```

(E)PROCESS?



Command - Local kernel - WinDbg:6.3.9600.17298 AMD64

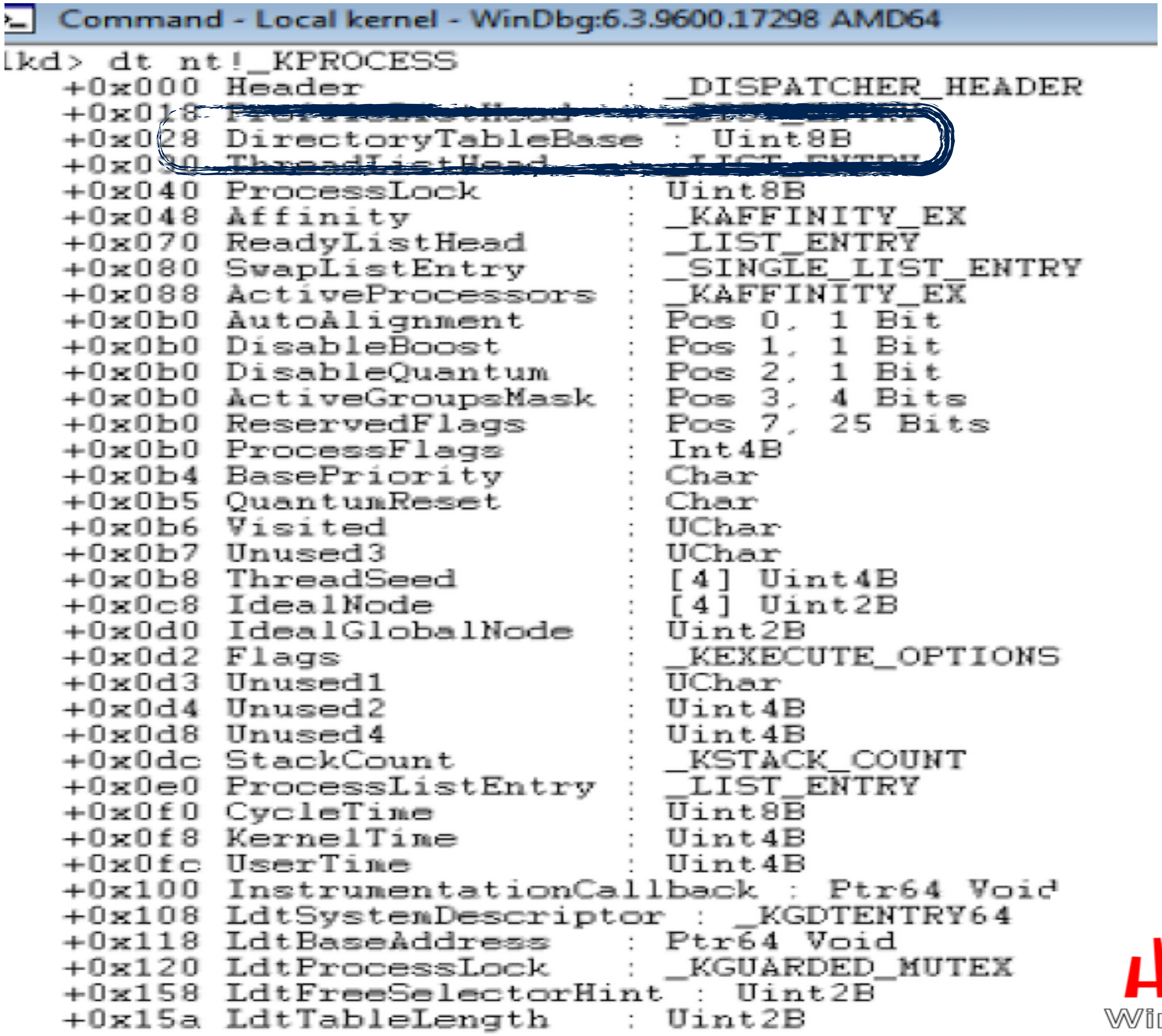
```
lkd> dt nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x160 ProcessLock      : _EX_PUSH_LOCK
   +0x168 CreateTime       : _LARGE_INTEGER
   +0x170 ExitTime         : _LARGE_INTEGER
   +0x178 RundownProtect   : _EX_RUNDOWN_REF
   +0x180 UniqueProcessId  : Ptr64 Void
   +0x188 ActiveProcessLinks : _LIST_ENTRY
   +0x198 ProcessQuotaUsage : [2] Uint8B
   +0x
   +0x
   +0x
   +0x
   +0x
   +0x
   +0x
   +0x1f0 DebugPort        : Ptr64 Void
   +0x1f8 ExceptionPortData : Ptr64 Void
   +0x1f8 ExceptionPortValue : Uint8B
   +0x1f8 ExceptionPortState : Pos 0, 3 Bits
   +0x200 ObjectTable      : Ptr64 _HANDLE_TABLE
   +0x208 Token            : _EX_FAST_REF
   +0x210 WorkingSetPage   : Uint8B
   +0x218 AddressCreationLock : _EX_PUSH_LOCK
   +0x220 RotateInProgress : Ptr64 _ETHREAD
   +0x228 ForkInProgress   : Ptr64 _ETHREAD
   +0x230 HardwareTrigger  : Uint8B
   +0x238 PhysicalVadRoot  : Ptr64 _MM_AVL_TABLE
   +0x240 CloneRoot        : Ptr64 Void
```



Command - Local kernel - WinDbg:6.3.9600.17298 AMD64

```
lkd> dt nt!_LIST_ENTRY
   +0x000 Flink            : Ptr64 _LIST_ENTRY
   +0x008 Blink            : Ptr64 _LIST_ENTRY
```

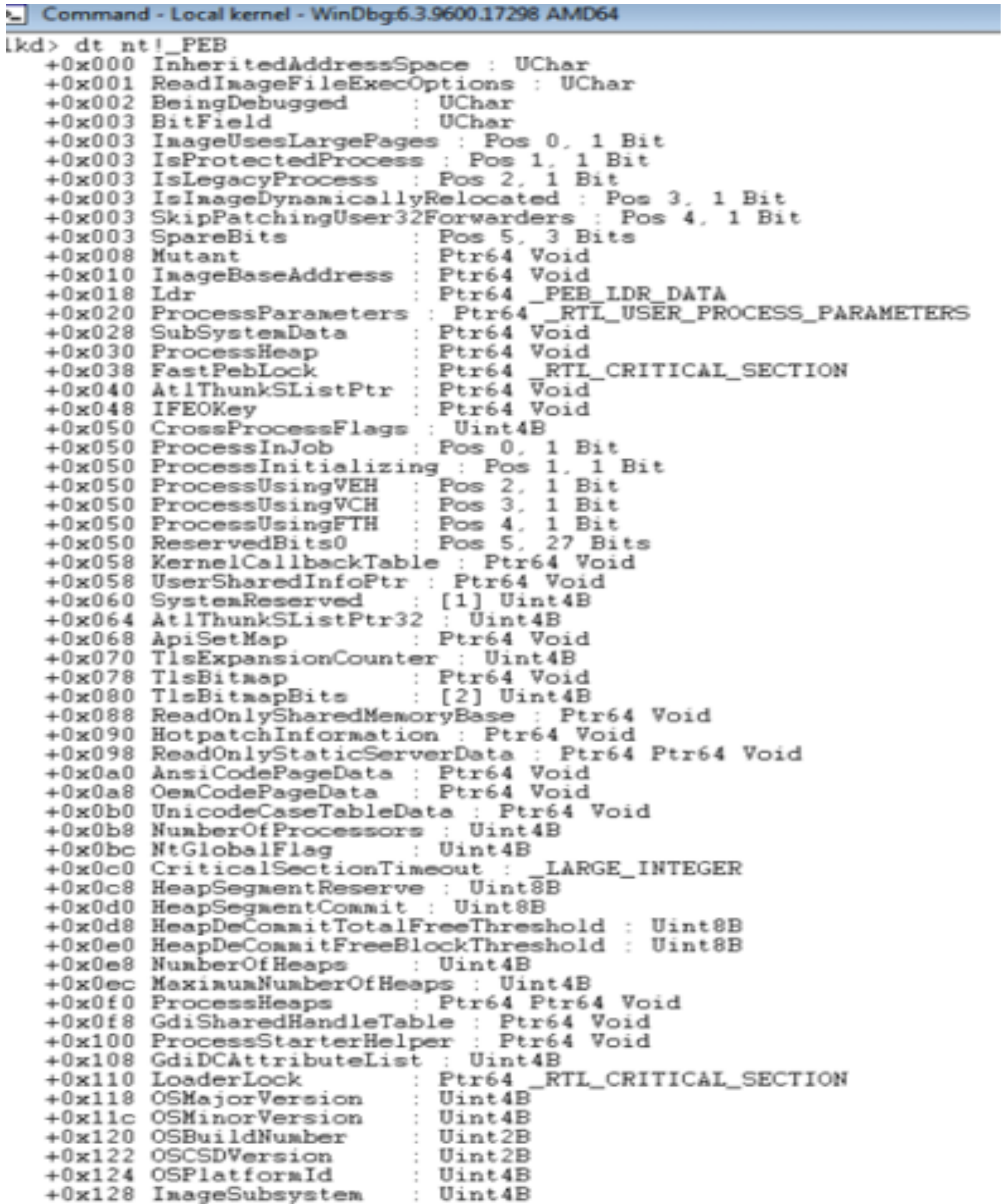
(K)PROCESS?



Command - Local kernel - WinDbg:6.3.9600.17298 AMD64

```
lkd> dt nt!_KPROCESS
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 ProfileListHead  : _LIST_ENTRY
   +0x028 DirectoryTableBase : Uint8B
   +0x030 ThreadListHead   : _LIST_ENTRY
   +0x040 ProcessLock      : Uint8B
   +0x048 Affinity         : _KAFFINITY_EX
   +0x070 ReadyListHead    : _LIST_ENTRY
   +0x080 SwapListEntry    : _SINGLE_LIST_ENTRY
   +0x088 ActiveProcessors : _KAFFINITY_EX
   +0x0b0 AutoAlignment    : Pos 0, 1 Bit
   +0x0b0 DisableBoost     : Pos 1, 1 Bit
   +0x0b0 DisableQuantum   : Pos 2, 1 Bit
   +0x0b0 ActiveGroupsMask : Pos 3, 4 Bits
   +0x0b0 ReservedFlags    : Pos 7, 25 Bits
   +0x0b0 ProcessFlags     : Int4B
   +0x0b4 BasePriority     : Char
   +0x0b5 QuantumReset     : Char
   +0x0b6 Visited          : UChar
   +0x0b7 Unused3          : UChar
   +0x0b8 ThreadSeed       : [4] Uint4B
   +0x0c8 IdealNode        : [4] Uint2B
   +0x0d0 IdealGlobalNode  : Uint2B
   +0x0d2 Flags            : _KEXECUTE_OPTIONS
   +0x0d3 Unused1          : UChar
   +0x0d4 Unused2          : Uint4B
   +0x0d8 Unused4          : Uint4B
   +0x0dc StackCount       : _KSTACK_COUNT
   +0x0e0 ProcessListEntry : _LIST_ENTRY
   +0x0f0 CycleTime        : Uint8B
   +0x0f8 KernelTime       : Uint4B
   +0x0fc UserTime         : Uint4B
   +0x100 InstrumentationCallback : Ptr64 Void
   +0x108 LdtSystemDescriptor : _KGDTENTRY64
   +0x118 LdtBaseAddress   : Ptr64 Void
   +0x120 LdtProcessLock   : _KGUARDED_MUTEX
   +0x158 LdtFreeSelectorHint : Uint2B
   +0x15a LdtTableLength   : Uint2B
```

PROCESS?



Command - Local kernel - WinDbg:6.3.9600.17298 AMD64

```
lkd> dt nt!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged    : UChar
   +0x003 BitField         : UChar
   +0x003 ImageUsesLargePages : Pos 0, 1 Bit
   +0x003 IsProtectedProcess : Pos 1, 1 Bit
   +0x003 IsLegacyProcess  : Pos 2, 1 Bit
   +0x003 IsImageDynamicallyRelocated : Pos 3, 1 Bit
   +0x003 SkipPatchingUser32Forwarders : Pos 4, 1 Bit
   +0x003 SpareBits        : Pos 5, 3 Bits
   +0x008 Mutant           : Ptr64 Void
   +0x010 ImageBaseAddress : Ptr64 Void
   +0x018 Ldr              : Ptr64 _PEB_LDR_DATA
   +0x020 ProcessParameters : Ptr64 _RTL_USER_PROCESS_PARAMETERS
   +0x028 SubSystemData    : Ptr64 Void
   +0x030 ProcessHeap      : Ptr64 Void
   +0x038 FastPebLock      : Ptr64 _RTL_CRITICAL_SECTION
   +0x040 AtlThunkSListPtr : Ptr64 Void
   +0x048 IFEOKey          : Ptr64 Void
   +0x050 CrossProcessFlags : Uint4B
   +0x050 ProcessInJob     : Pos 0, 1 Bit
   +0x050 ProcessInitializing : Pos 1, 1 Bit
   +0x050 ProcessUsingVEH  : Pos 2, 1 Bit
   +0x050 ProcessUsingVCH  : Pos 3, 1 Bit
   +0x050 ProcessUsingFTH  : Pos 4, 1 Bit
   +0x050 ReservedBits0    : Pos 5, 27 Bits
   +0x058 KernelCallbackTable : Ptr64 Void
   +0x058 UserSharedInfoPtr : Ptr64 Void
   +0x060 SystemReserved   : [1] Uint4B
   +0x064 AtlThunkSListPtr32 : Uint4B
   +0x068 ApiSetMap        : Ptr64 Void
   +0x070 TlsExpansionCounter : Uint4B
   +0x078 TlsBitmap        : Ptr64 Void
   +0x080 TlsBitmapBits    : [2] Uint4B
   +0x088 ReadOnlySharedMemoryBase : Ptr64 Void
   +0x090 HotpatchInformation : Ptr64 Void
   +0x098 ReadOnlyStaticServerData : Ptr64 Ptr64 Void
   +0x0a0 AnsiCodePageData : Ptr64 Void
   +0x0a8 OemCodePageData  : Ptr64 Void
   +0x0b0 UnicodeCaseTableData : Ptr64 Void
   +0x0b8 NumberOfProcessors : Uint4B
   +0x0bc NtGlobalFlag     : Uint4B
   +0x0c0 CriticalSectionTimeout : _LARGE_INTEGER
   +0x0c8 HeapSegmentReserve : Uint8B
   +0x0d0 HeapSegmentCommit : Uint8B
   +0x0d8 HeapDeCommitTotalFreeThreshold : Uint8B
   +0x0e0 HeapDeCommitFreeBlockThreshold : Uint8B
   +0x0e8 NumberOfHeaps    : Uint4B
   +0x0ec MaximumNumberOfHeaps : Uint4B
   +0x0f0 ProcessHeaps     : Ptr64 Ptr64 Void
   +0x0f8 GdiSharedHandleTable : Ptr64 Void
   +0x100 ProcessStarterHelper : Ptr64 Void
   +0x108 GdiDCAttributeList : Uint4B
   +0x110 LoaderLock       : Ptr64 _RTL_CRITICAL_SECTION
   +0x118 OSMajorVersion   : Uint4B
   +0x11c OSMinorVersion   : Uint4B
   +0x120 OSBuildNumber    : Uint2B
   +0x122 OSCSDVersion     : Uint2B
   +0x124 OSPlatformId     : Uint4B
   +0x128 ImageSubsystem   : Uint4B
```

HACK IN BO Winter 2016 Edition

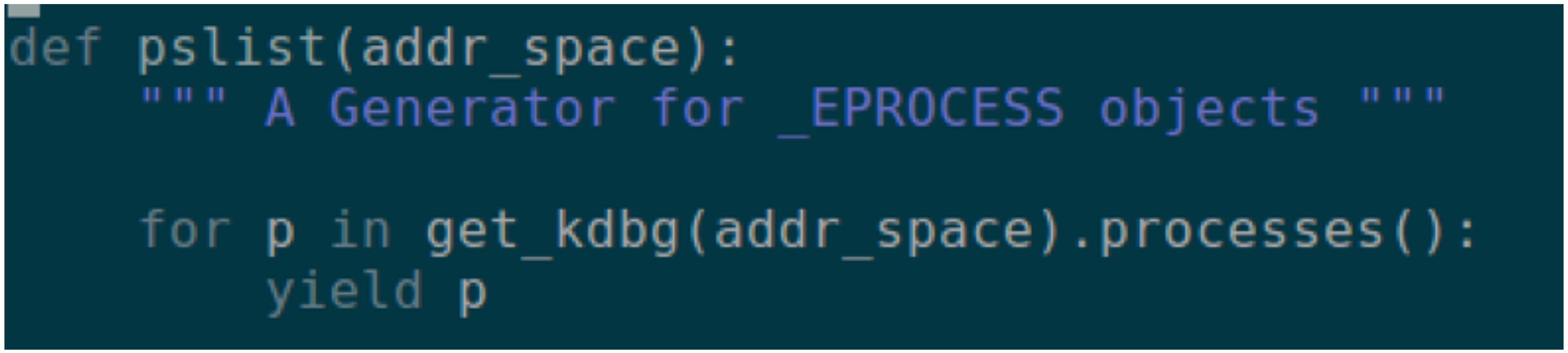
PROCESS?

- EPROCESS info:
  - Creation and exit time
  - PID and PPID
  - Pointer to the handle table
  - VAD, etc
- PEB info:
  - Pointer to the Image Base Address
  - Pointer to the DLLs loaded
  - Heap size, etc

DKOM DEFENSES

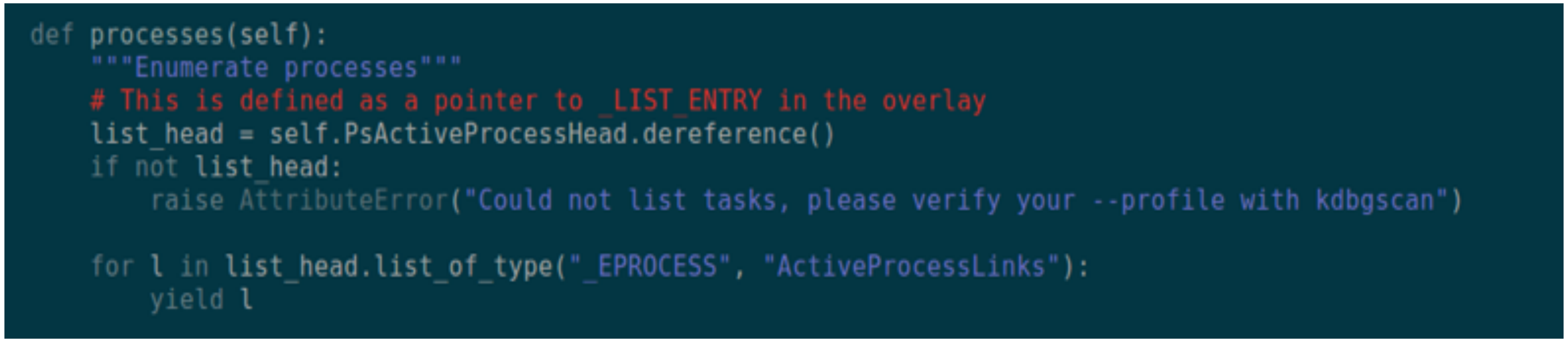
- Kernel data integrity solutions:
    - invariants
        - external systems
        - memory analysis
    - data partitioning

VOLATILITY - PSLIST



```
def pslist(addr_space):
    """ A Generator for _EPROCESS objects """

    for p in get_kdbg(addr_space).processes():
        yield p
```



```
def processes(self):
    """Enumerate processes"""
    # This is defined as a pointer to _LIST_ENTRY in the overlay
    list_head = self.PsActiveProcessHead.dereference()
    if not list_head:
        raise AttributeError("Could not list tasks, please verify your --profile with kdbgscan")

    for l in list_head.list_of_type("_EPROCESS", "ActiveProcessLinks"):
        yield l
```

DEMO

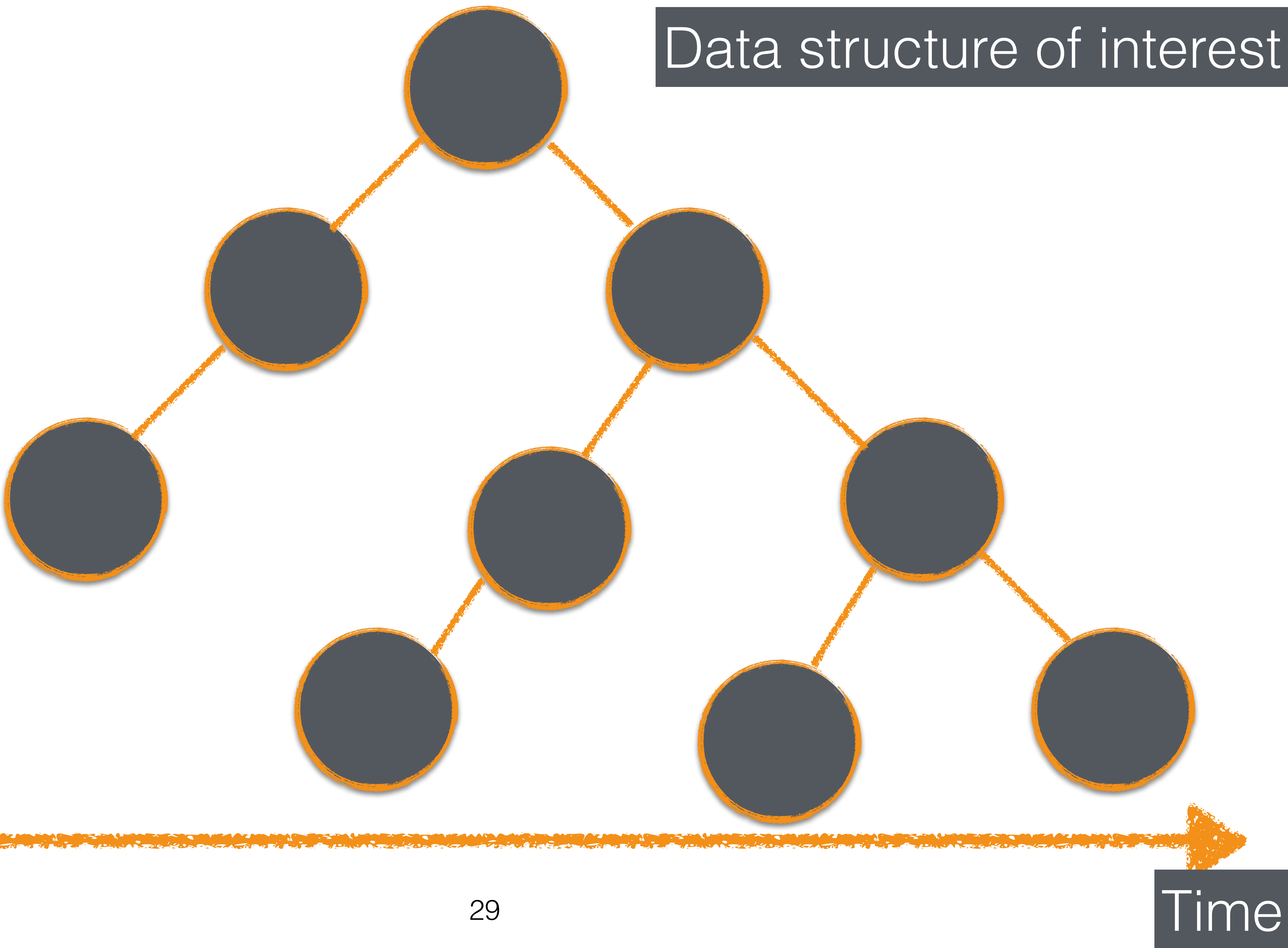
“DKOM DEMO”

E-DKOM

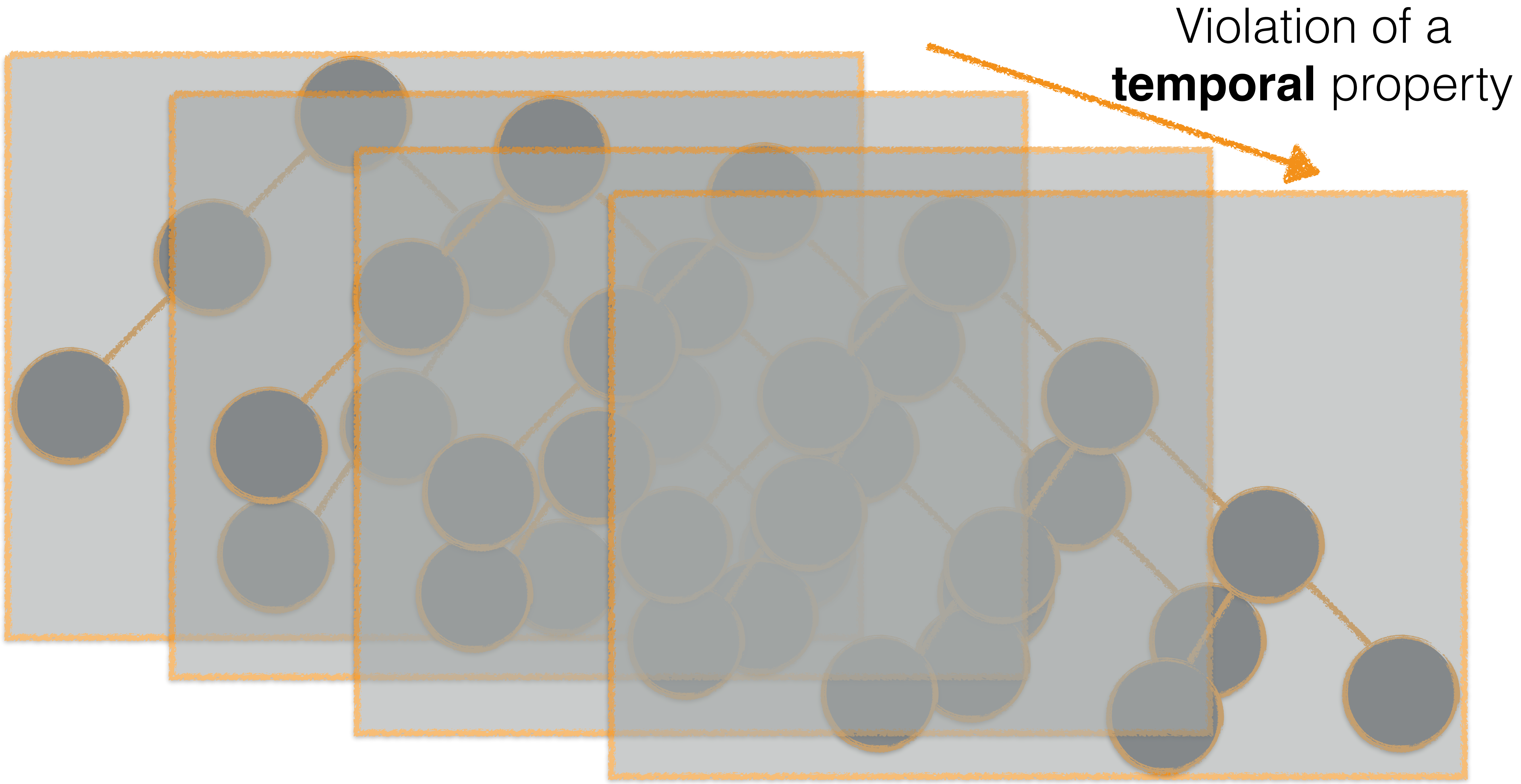
"**E**volutionary **D**irect **K**ernel **O**bject **M**anipulation"

"Subverting Operating System Properties through Evolutionary DKOM Attacks"
Mariano Graziano, Lorenzo Flore, Andrea Lanzi, Davide Balzarotti
DIMVA 2016, San Sebastian, Spain

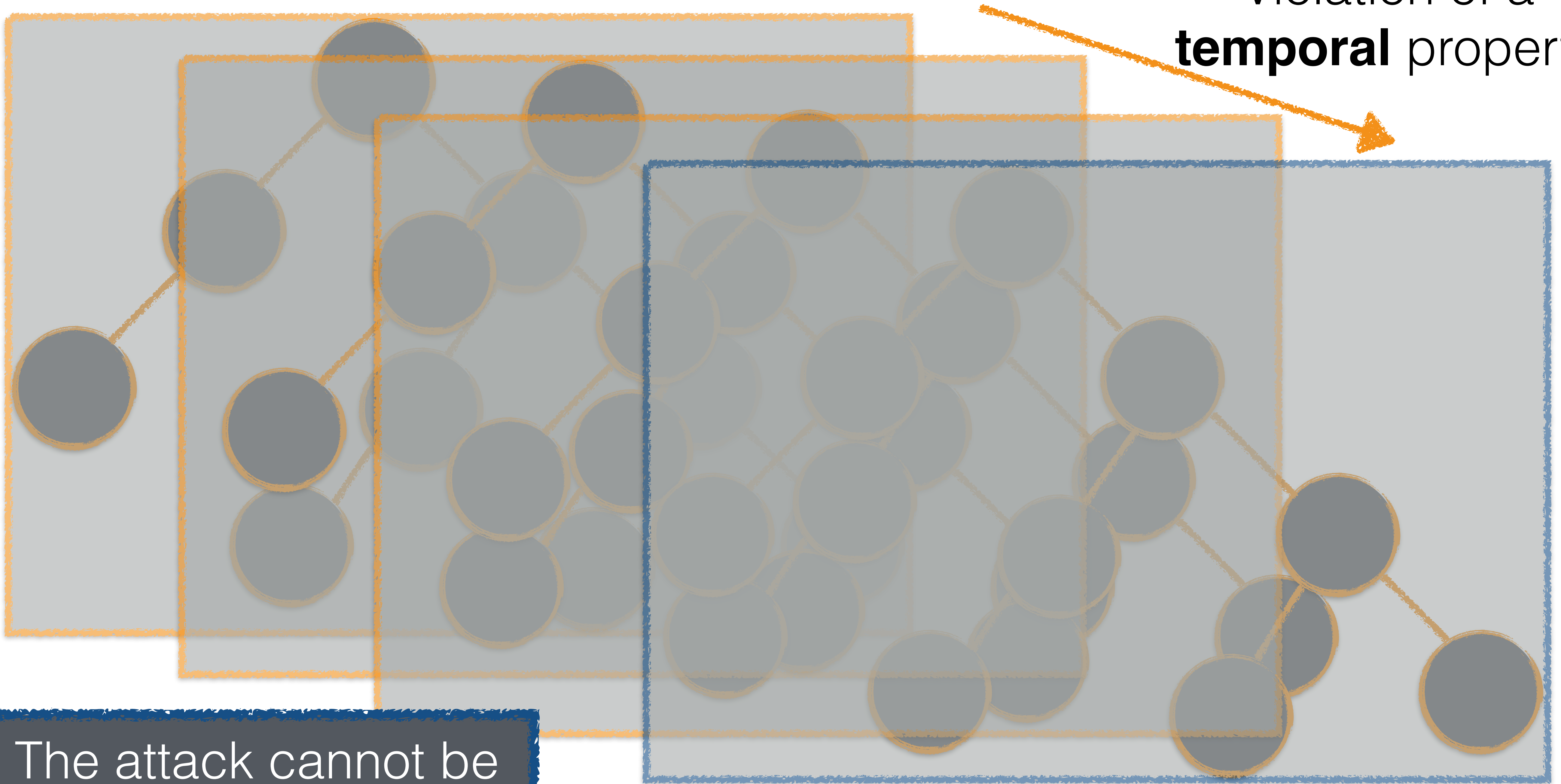
E-DKOM



E-DKOM



E-DKOM



The attack cannot be detected looking at a single snapshot

STATE vs PROPERTY

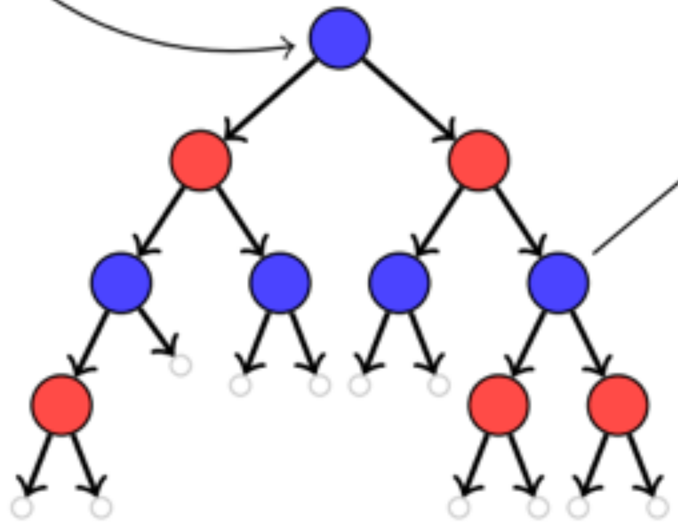
- Traditional DKOM affects the state and are discrete
- Evolutionary DKOM (E-DKOM) affects the evolution in time of a given property and are continuous

LINUX CFS SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```



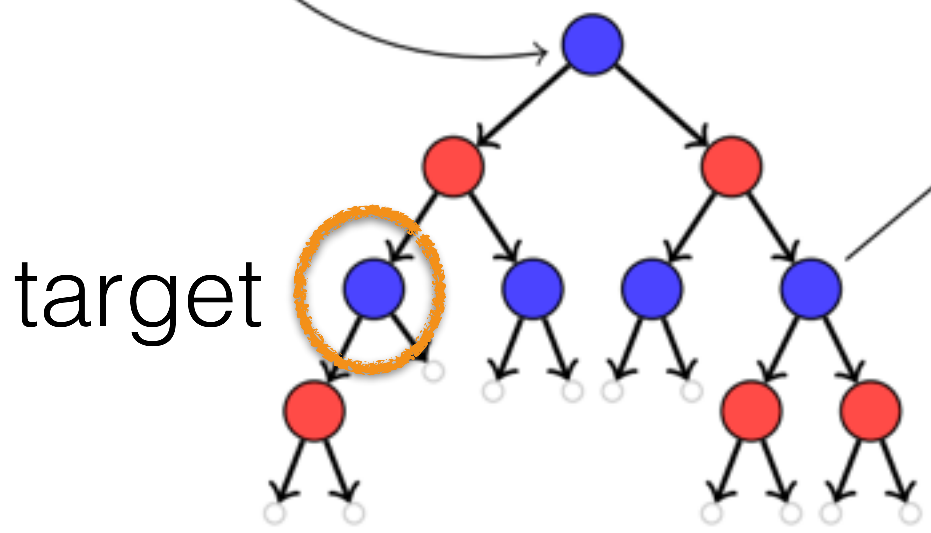
```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```

LINUX CFS SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```



```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```

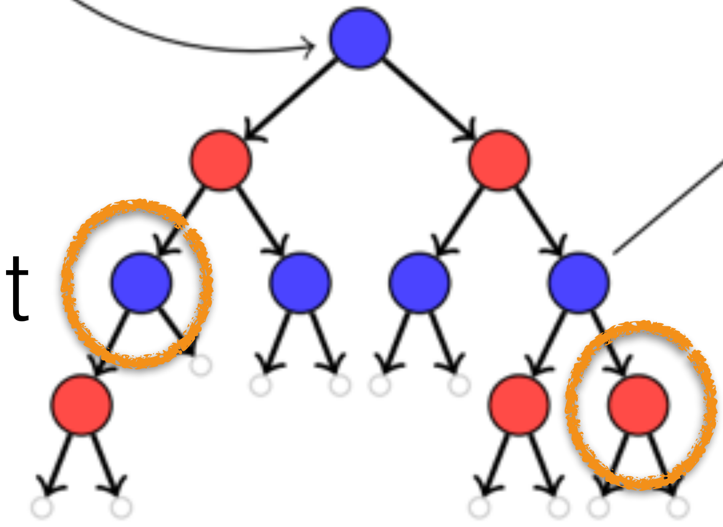
LINUX CFS SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

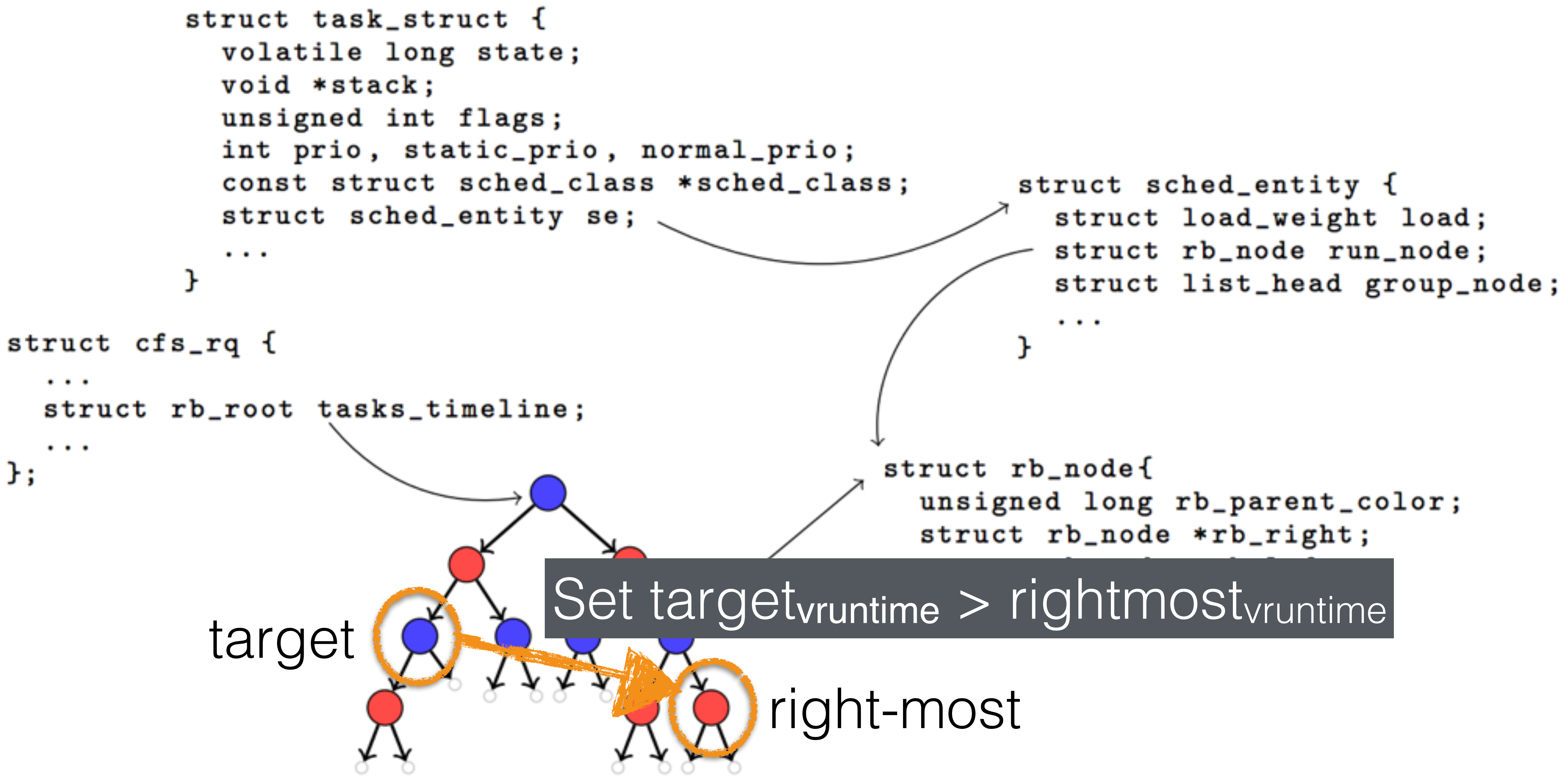
```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```



LINUX CFS SCHEDULER



```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
```

Set $\text{target}_{\text{vruntime}} > \text{rightmost}_{\text{vruntime}}$

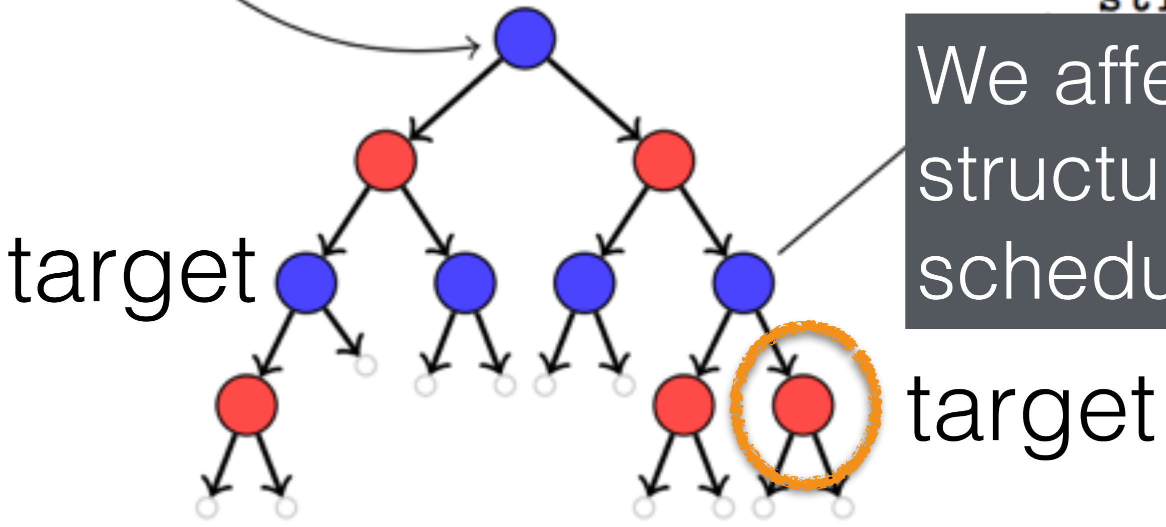
LINUX CFS SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node{
```



We affect the **evolution** of the data structure over time. We altered the scheduler **property** (fair execution)

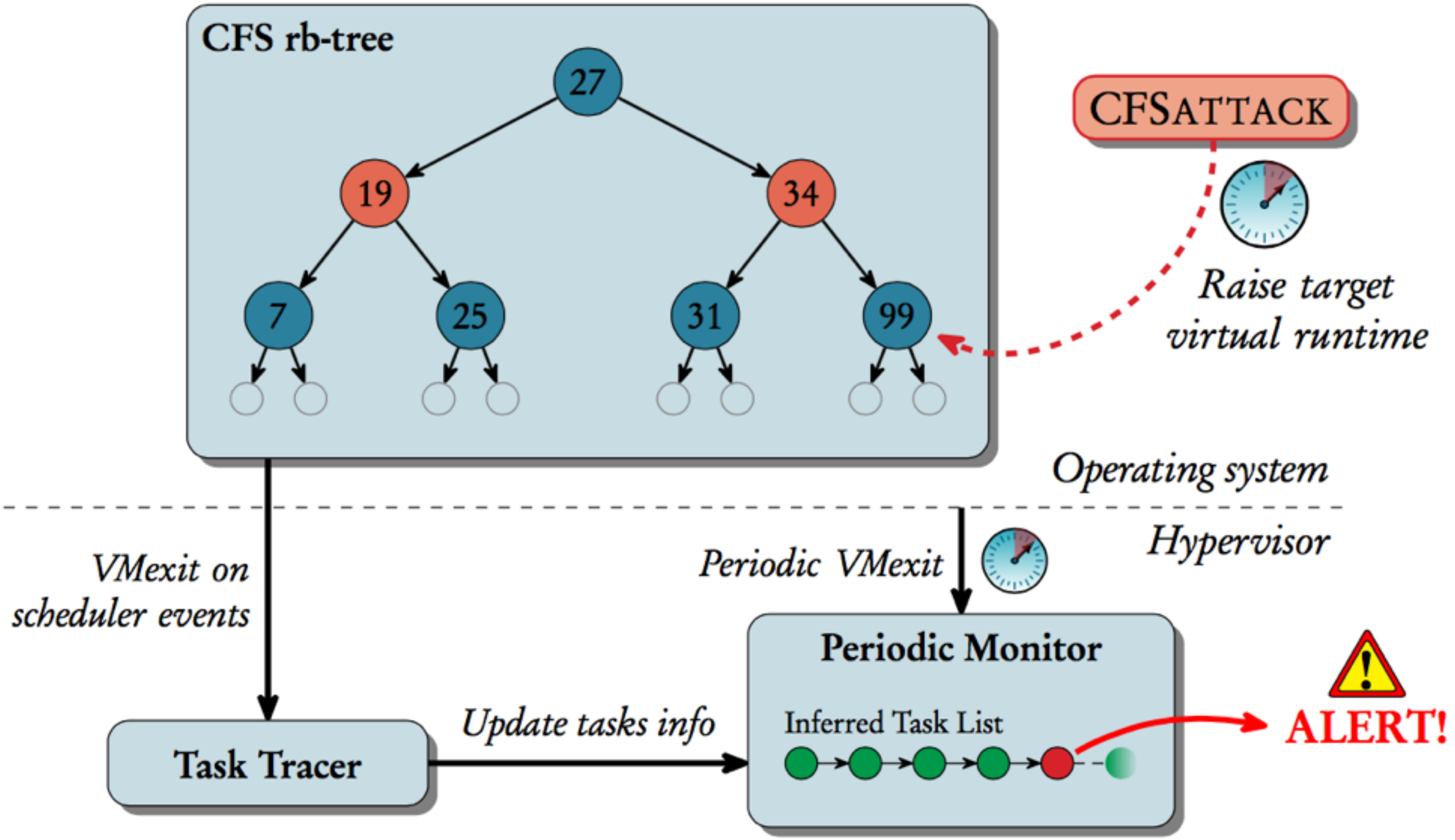
DEMO

"E-DKOM DEMO"

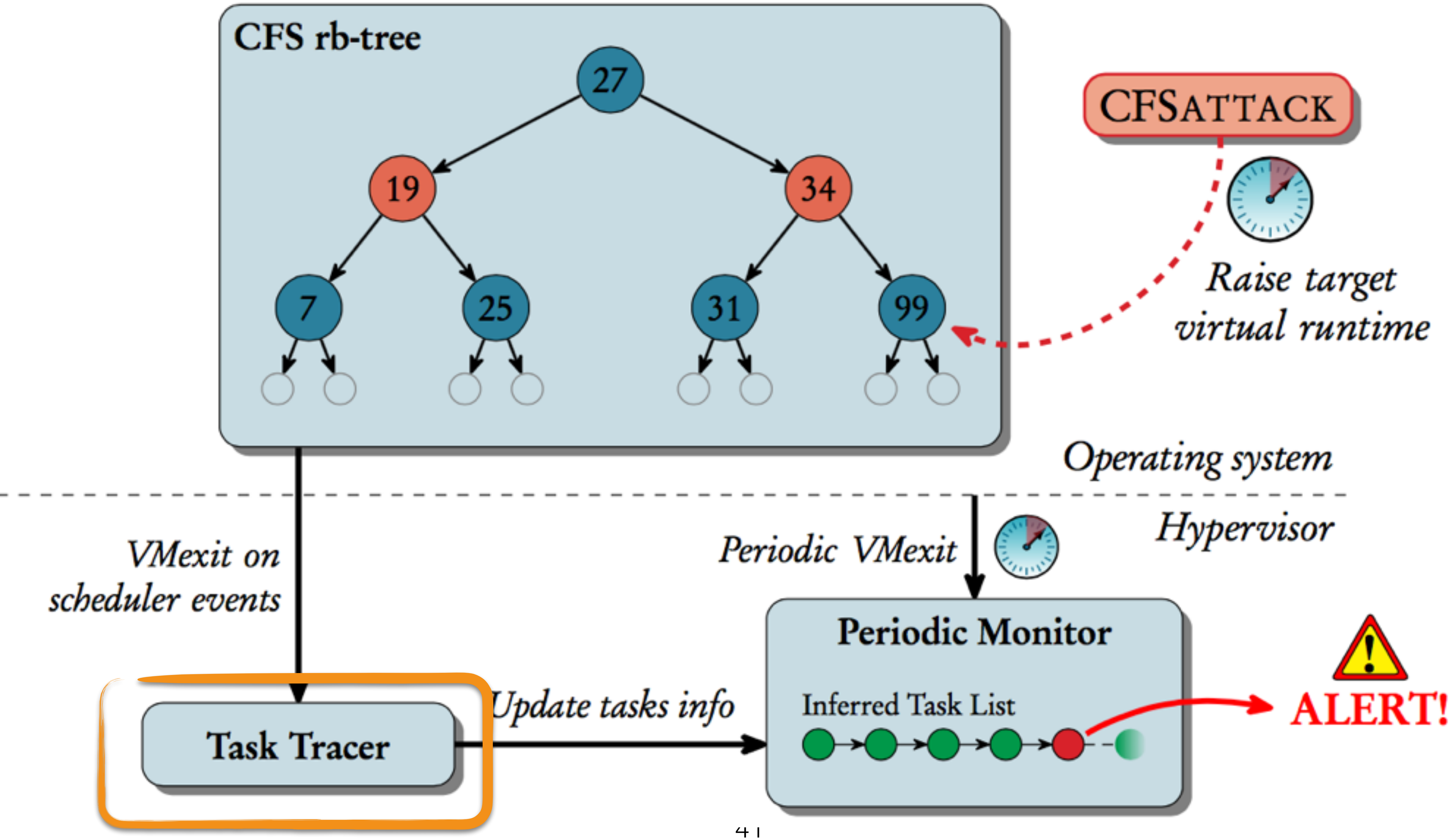
DEFENSES?

- Reference monitor that mimics the OS property:
    - OS specific
    - Difficult to generalize

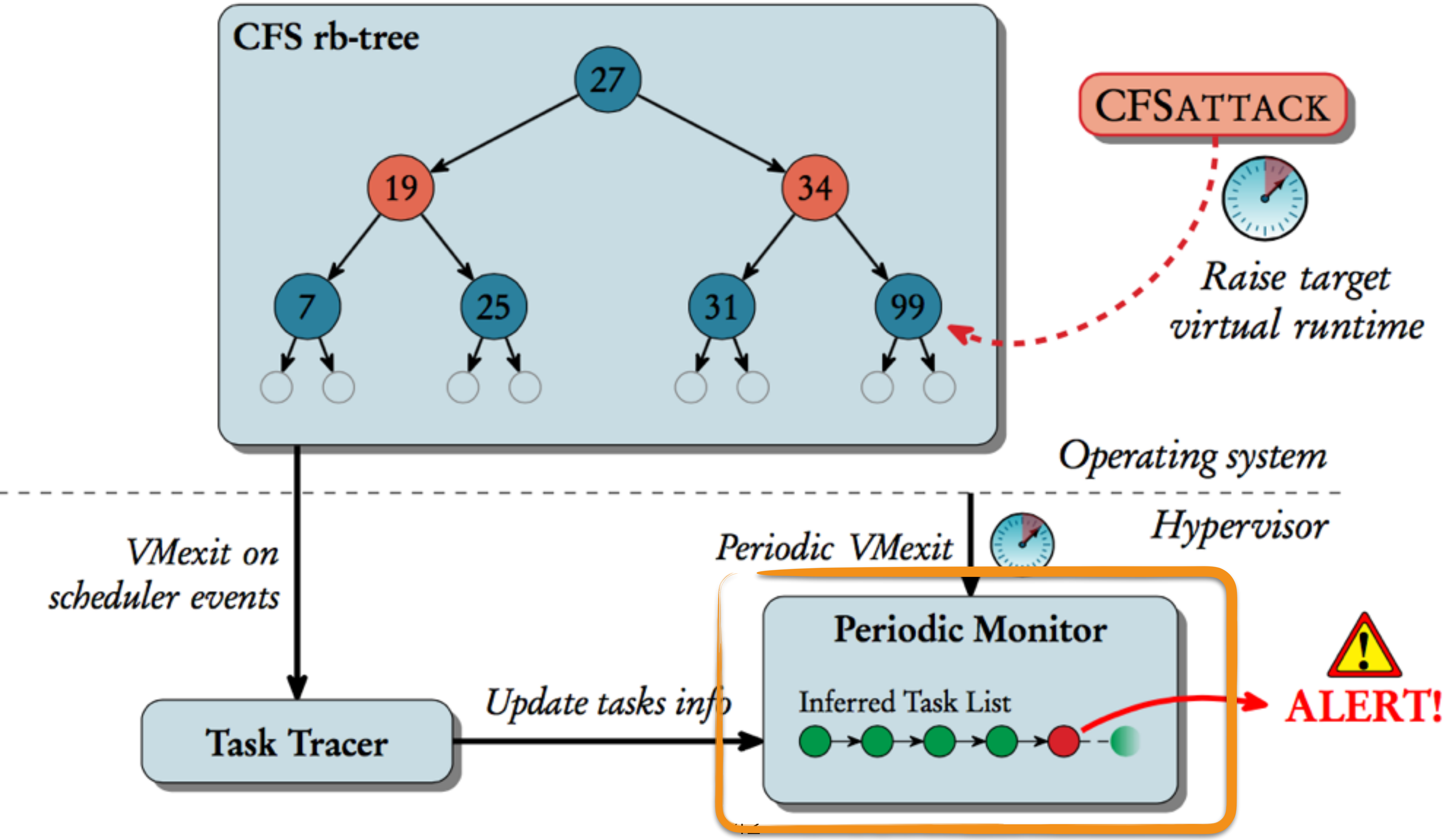
DEFENSE FRAMEWORK



DEFENSE FRAMEWORK



DEFENSE FRAMEWORK



FUTURE

- Minimalism
- Possibile trends:
    - Infections for the masses
    - Stealthy and multi stage attacks
- Cat and mouse game
- Microsoft approach:
    - Credential Guard
    - Application Guard

CONCLUSION

- Rootkit technology evolution
- New attack based on data structure evolution
- Experiment on the Linux CFS scheduler
- Defense based on hypervisor
- General mitigation/solution very hard

THE END

THANK YOU

email: magrazia@cisco.com
twitter: @emd3l