Rootkits

Using memory dumps to detect them

Disclaimer

I am a pony

All credits for this work belong to the authors of the “references” link

Rootkit: What?

- Keep root access
- 14 years, 2000 : he4hook, HacDef, Vanquish

Rootkit: types

- **Ring 3** **User mode**
- **Ring 0** **Kernel mode**
- Ring -1 Hypervisor mode
- Ring -2 System Management Mode
- Ring -3 Active Management Technology (*Blackhat creeps...*)

Rootkit: Why?

- Illegal :
  - Trojans
  - Customizable
  - Custom, targeted attacks
- Legal?
  - SONY: Digital Rights Management, prevent copies
  - Symantec, ...

Rootkit: How

Persistent rootkits:

- Startup files
- Registry keys
- Add-on to existing application
- Patching bootloader, kernel, driver
- Custom Master Boot Record
- BIOS

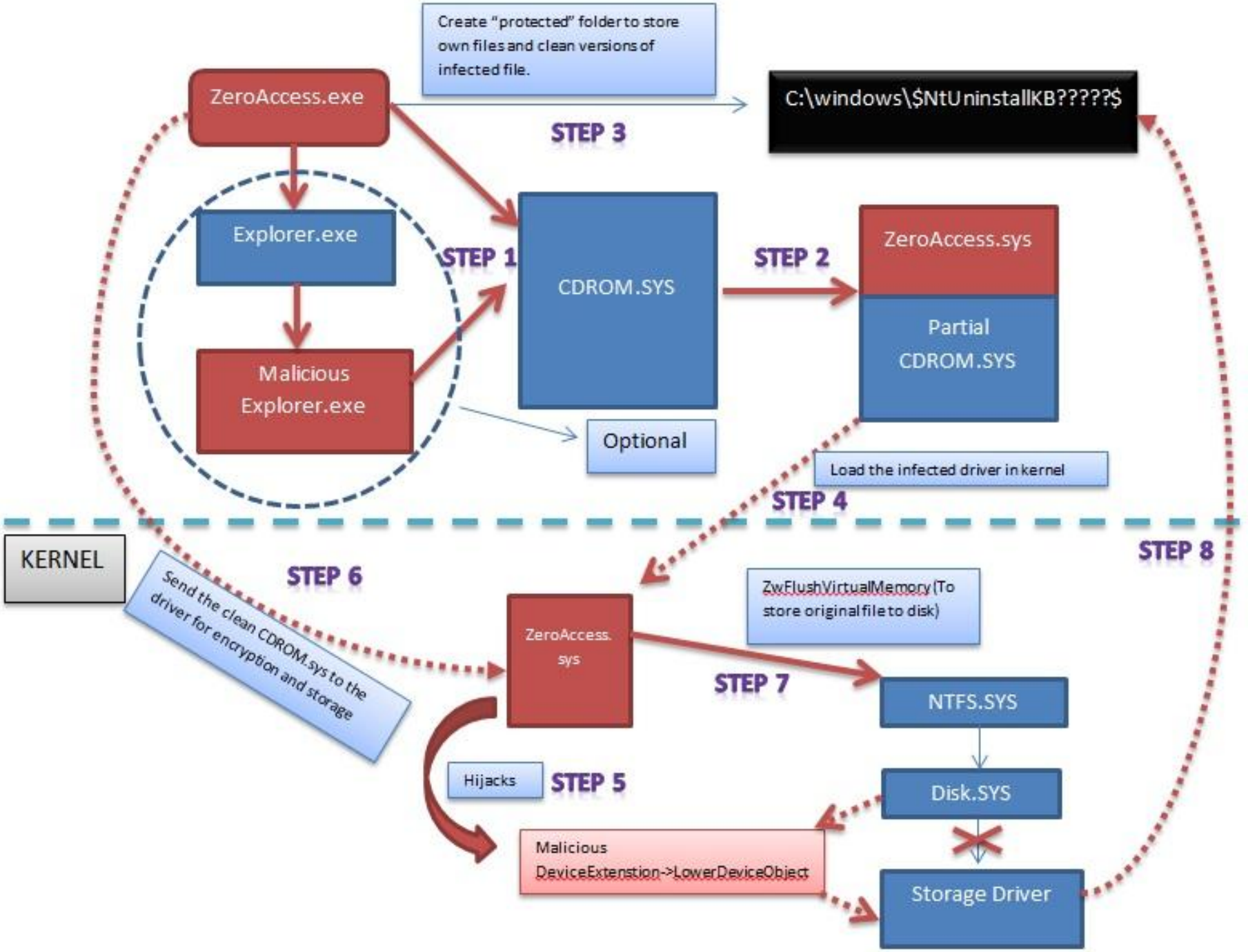
Rootkit: User mode

User mode rootkits:

- int 2Eh / sysenter ⇔ syscall : ring3 -> ring0
- LD_PRELOAD

Rootkit: Kernel mode

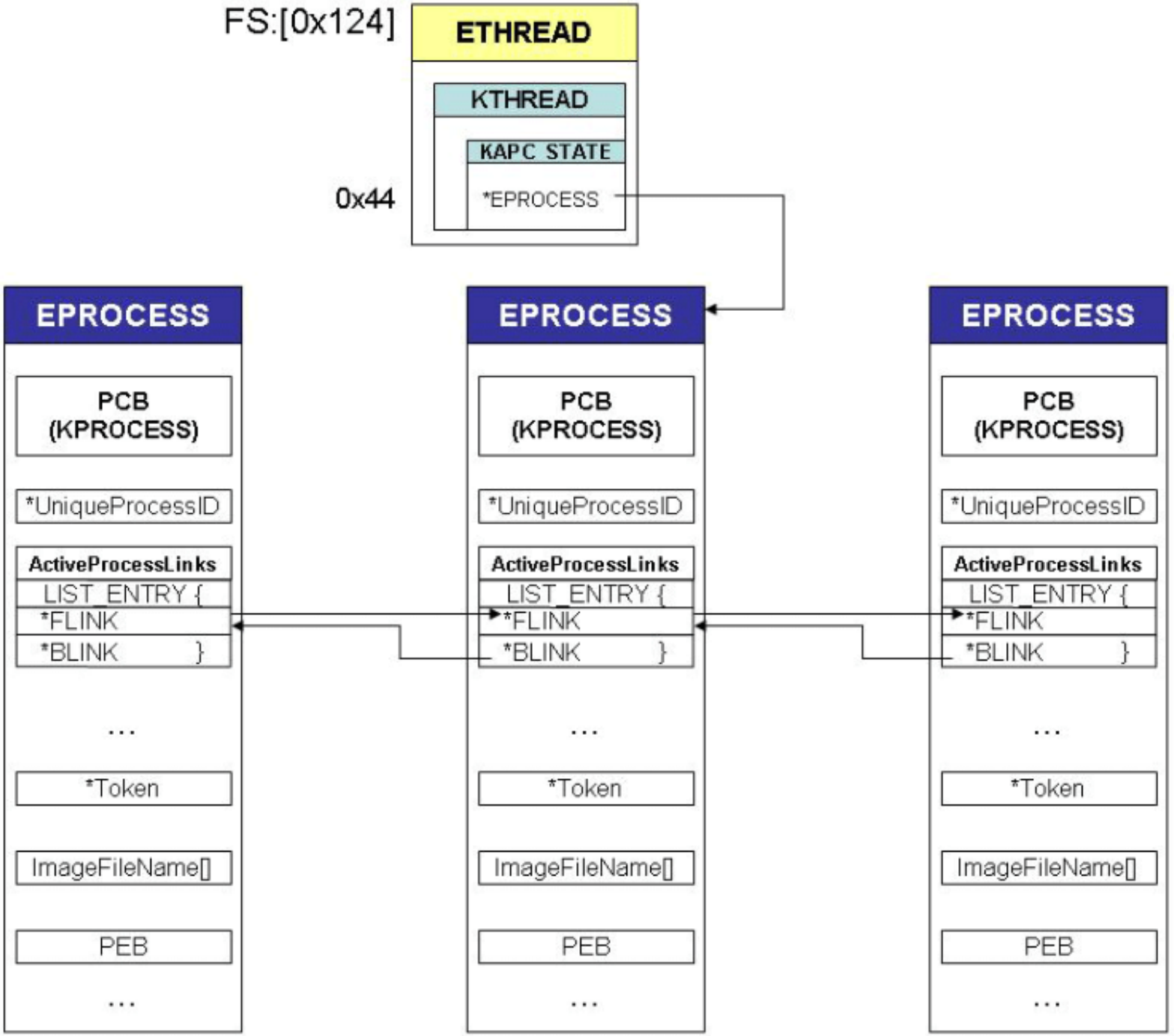
Hooking + injecting

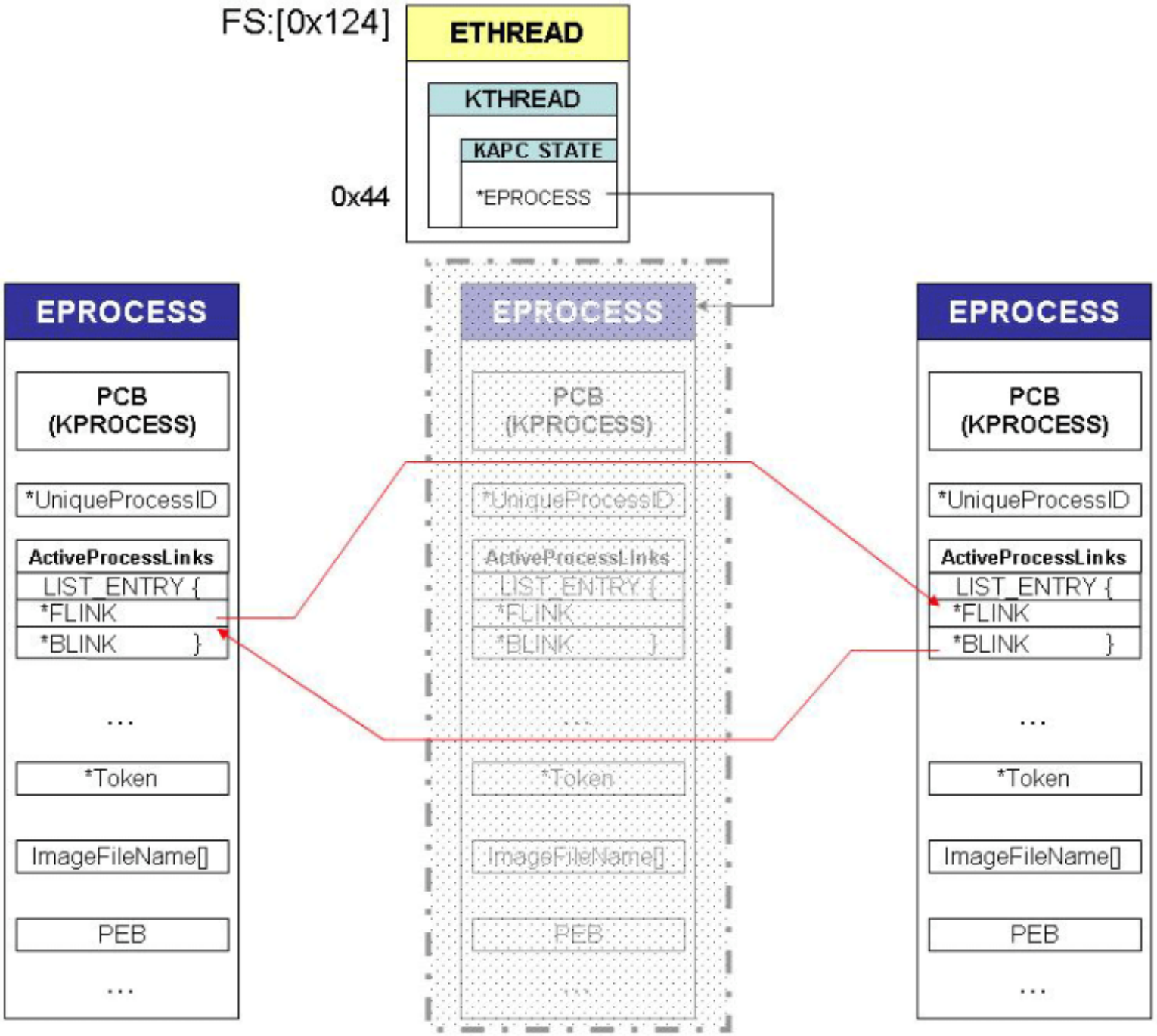

Create "protected" folder to store own files and clean versions of infected file.
ZeroAccess.exe
C:\windows\$NtUninstallKB?????$
STEP 3
Explorer.exe
STEP 1
CDROM.SYS
STEP 2
ZeroAccess.sys
Partial CDROM.SYS
Malicious Explorer.exe
Optional
Load the infected driver in kernel
STEP 4
KERNEL
STEP 8
STEP 6
Send the clean CDROM.sys to the driver for encryption and storage
ZeroAccess. sys
ZwFlushVirtualMemory (To store original file to disk)
STEP 7
NTFS.SYS
Hijacks
STEP 5
Disk.SYS
Malicious DeviceExtenstion->LowerDeviceObject
Storage Driver

Rootkit: Kernel mode

Unlinking :

Direct Kernel Object Manipulation (DKOM)


FS:[0x124]
ETHREAD
KTHREAD
KAPC STATE
0x44
*EPROCESS
EPROCESS
PCB
(KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB
(KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB
(KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...


FS:[0x124]
ETHREAD
KTHREAD
KAPC STATE
0x44
*EPROCESS
EPROCESS
PCB (KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB (KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB (KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...

Rootkit: Detection methods

- Signatures checks
- Heuristics checks
- Monitoring interceptions
- Data comparison (corruption of high-level API, amount of RAM used compared to what is stored)
- Integrity checks (*sums)

Rootkit: Memory dumps

- Crash dump
- Raw dump
- hyberfile/pagefile.sys
- Vmem

Rootkit: Redirect tables

- Interrupt Descriptor Table (interrupts)
- System Service Dispatch Table (syscalls)
- Import Address Table (external calls)

Rootkit: Interrupt

Event requiring attention NOW.

Scheduler switch execution “context”

Rootkit: Windbg

- Applications
- Drivers
- Kernel

Rootkit: Windbg

IDT:

System interrupt : int 2e
System calls : KiSystemService

$> !idt 2e == &KiSystemService

Rootkit: Windbg

List differences between syscalls codes and their copies :

$> !chkimg -d nt

Rootkit: Windbg

Hidden modules:

PE Format:

4d5a9000 -> 'MZ\x90\x00'

$> s -d 0x0 L?0xffffffff 0x00905a4d

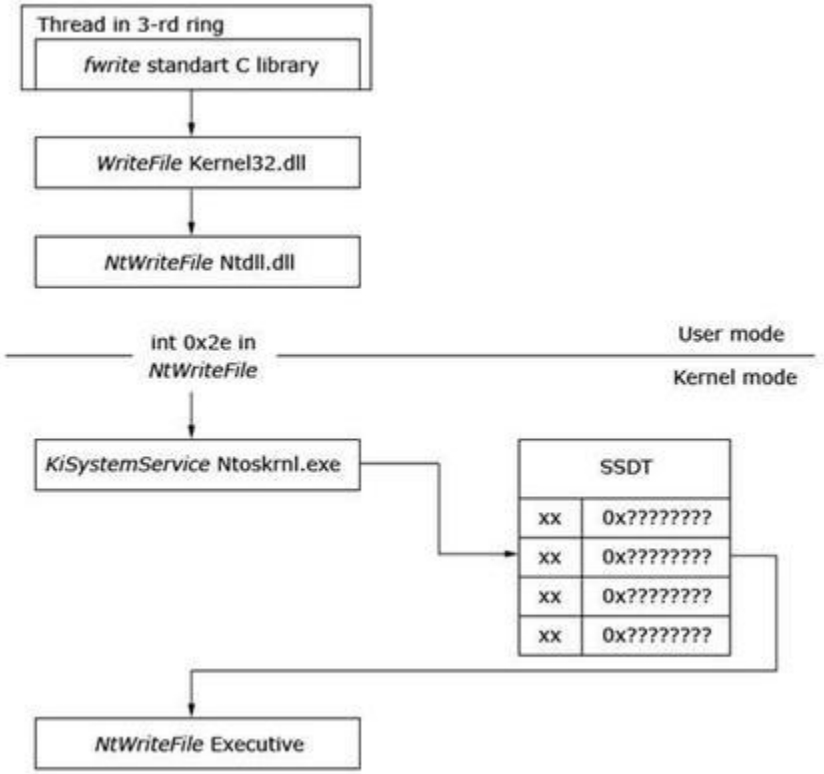
01000000 00905a4d 00000003 00000004 0000ffff MZ.....

Display module info if module was properly loaded:

$> !lmi ffffffff883ecc0

ffffffff883ecc0 is not a valid address

Rootkit: SSDT



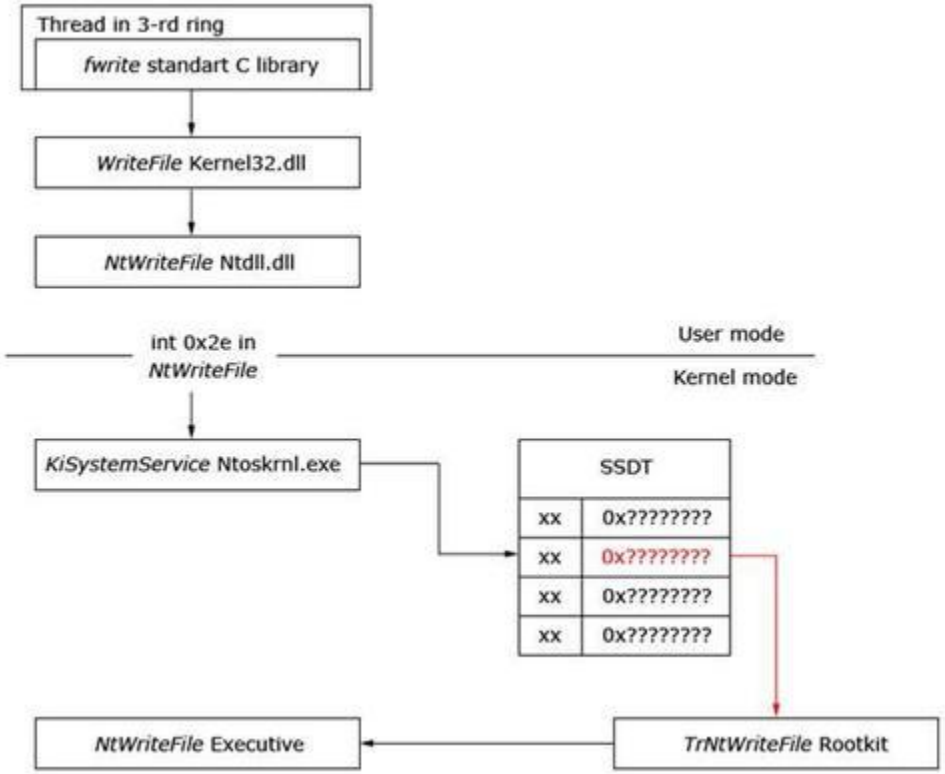
RootKit: SSDT

```
PVOID HookSystemCall(PVOID SystemCallFunction, PVOID HookFunction)
{
  ULONG SystemCallIndex = *(ULONG *)((PCHAR)SystemCallFunction+1);
  PVOID *NativeSystemCallTable = KeServiceDescriptorTable[0];
  PVOID OriginalSystemCall = NativeSystemCallTable[SystemCallIndex];

  NativeSystemCallTable[SystemCallIndex] = HookFunction;

  return OriginalSystemCall;
}
```

Rootkit: SSDT



RootKit: SSDT

SSDT pointers must refer to routines in *nt* or *win32k* library

Threat detection : Belkasoft

RootKit: Threads services hook

SSDT, not you again!

KTHREAD.pServiceDescriptorTable:

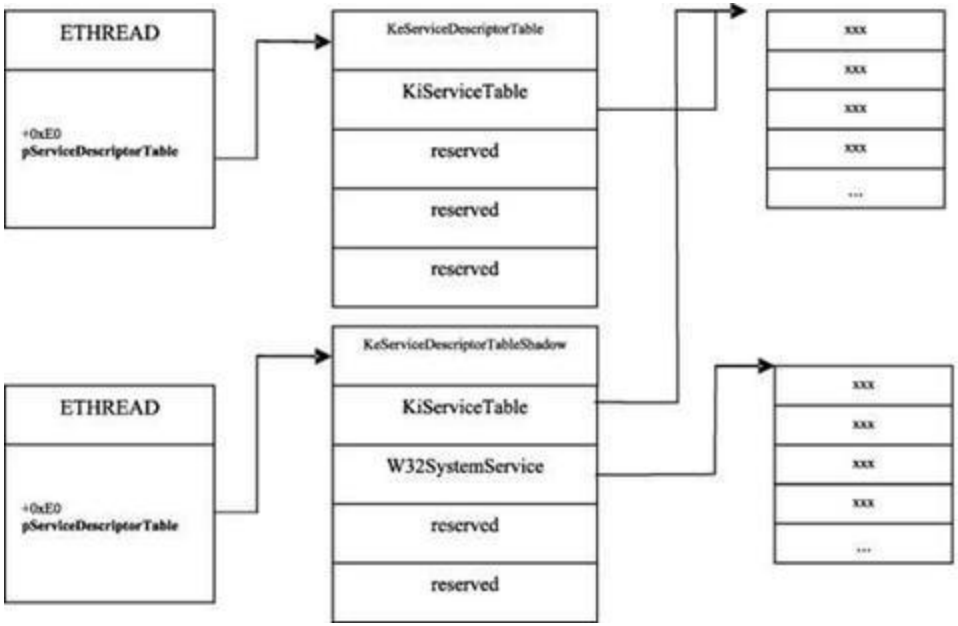
A thread can modify its own SSDT

RootKit: Threads services hook

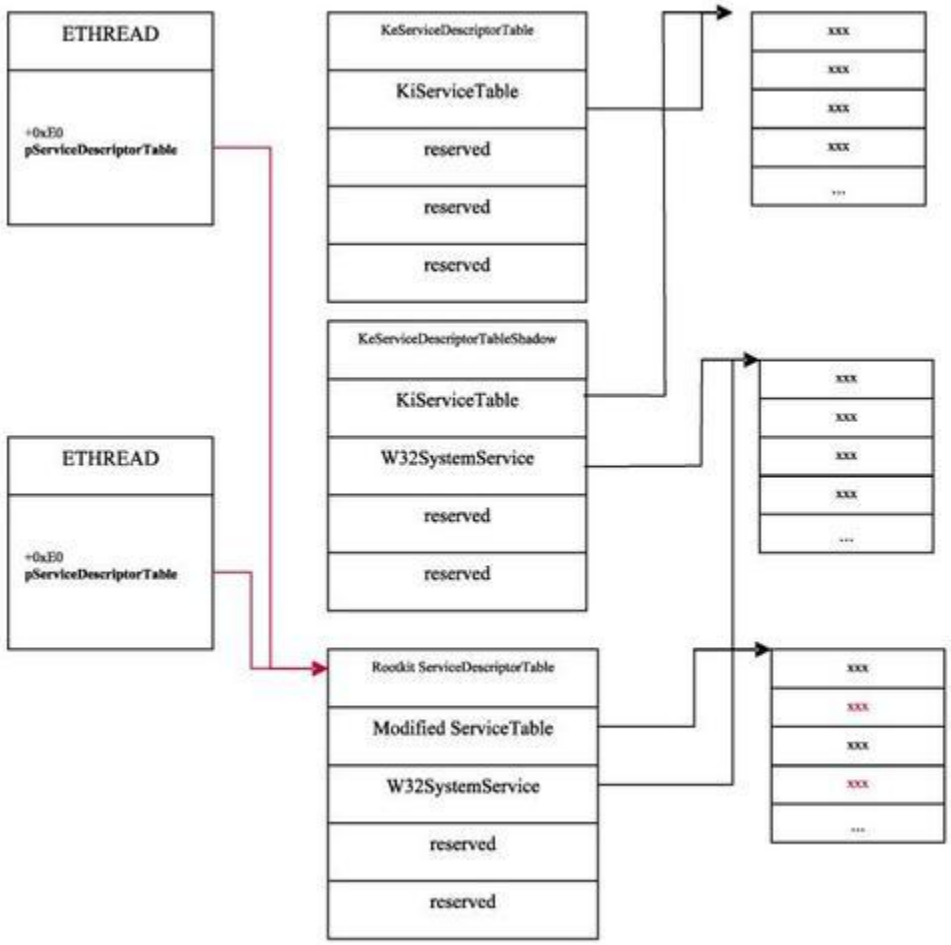
Actual functions pointers must be matched against those in :

- nt!KeServiceDescriptorTable
- nt!KeServiceDescriptorTableShadow

RootKit: Threads services hook



RootKit: Threads services hook

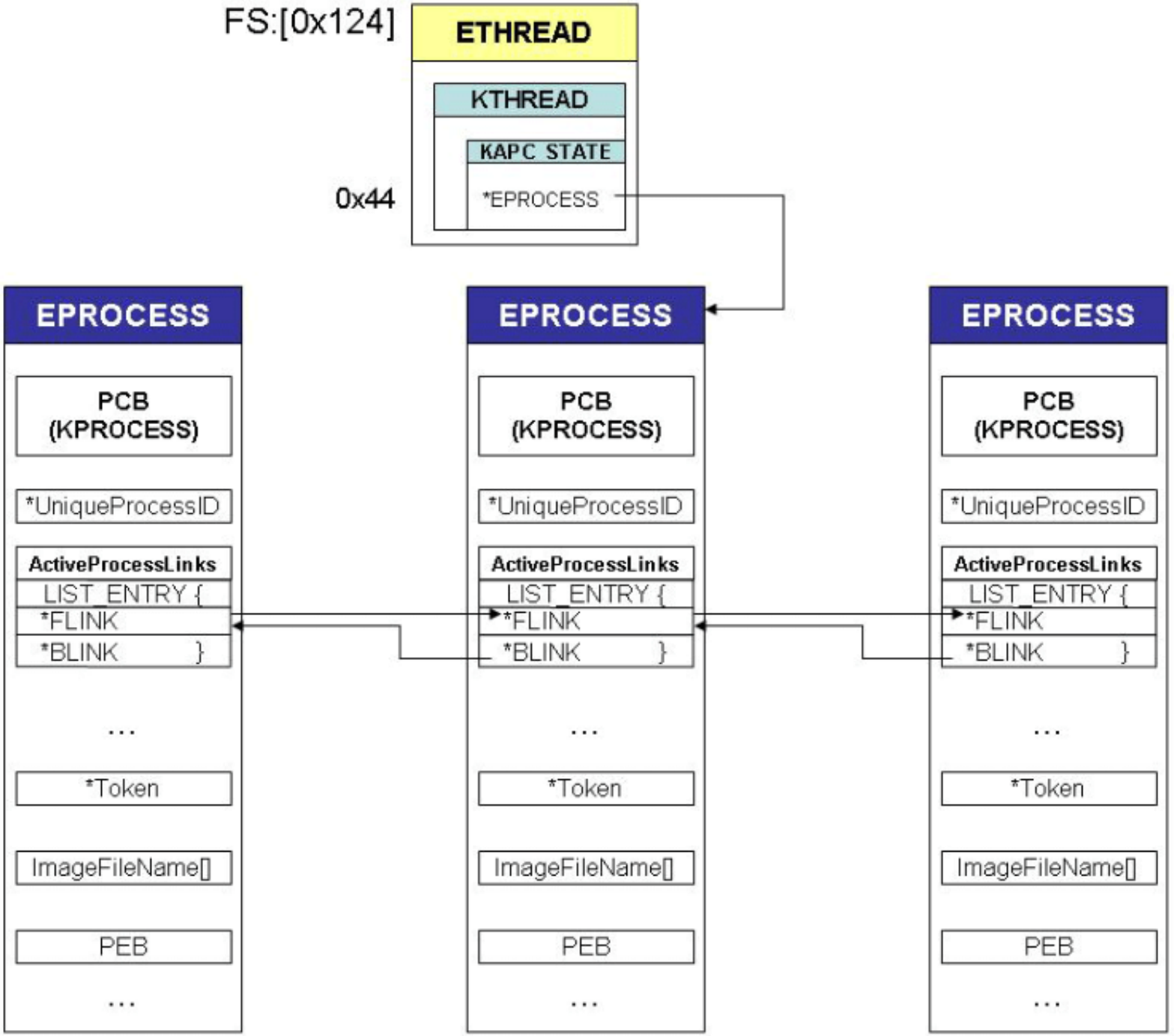


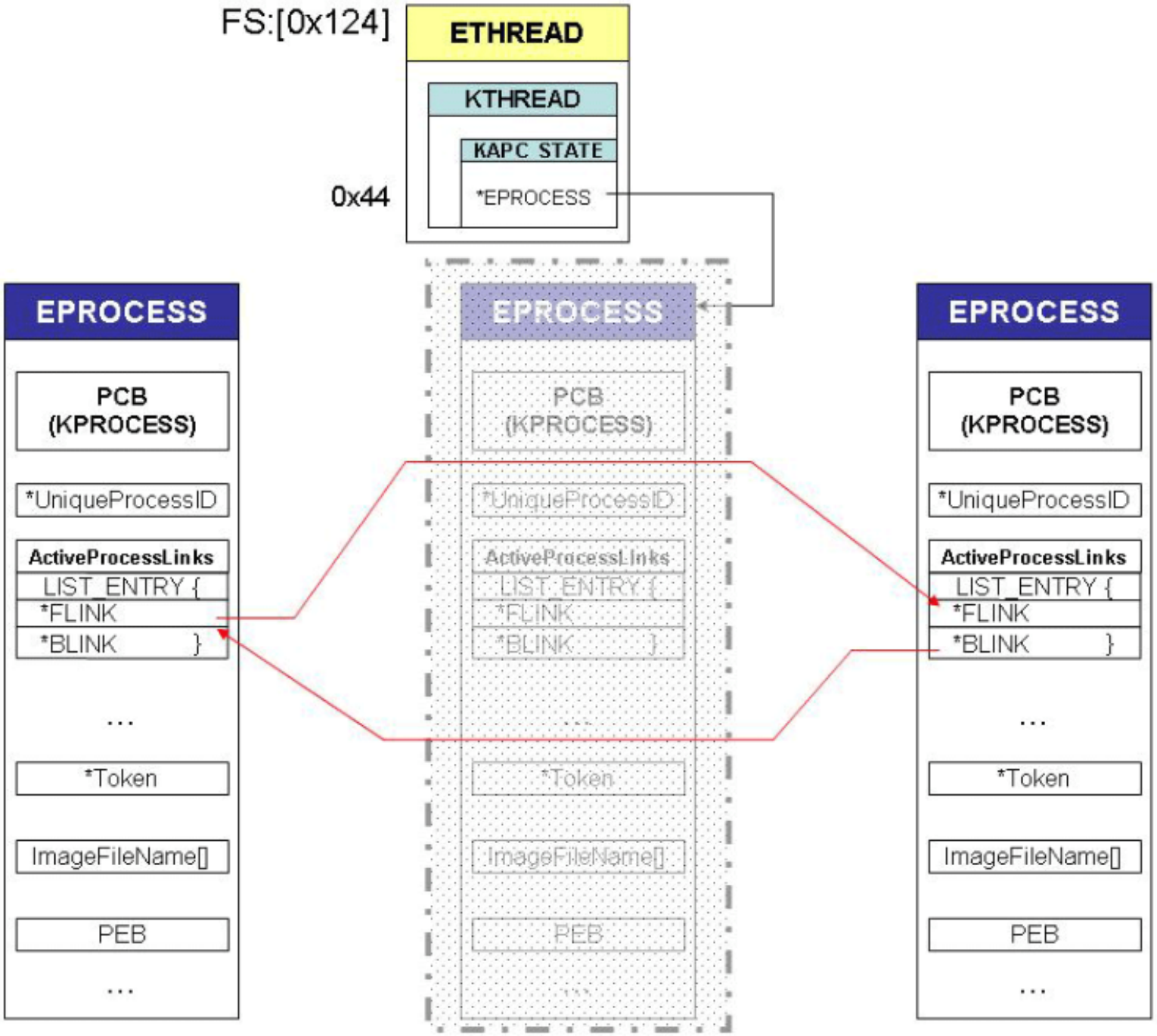
RootKit: Hooking tables

- GDT (Global Descriptor Table)
- LDT (Local Descriptor Table)
- EAT (Export Address Table)
- IAT (Import Address Table)
- IRP (I/O Request Packet)

Rootkit: Hidden process

Remember the process hiding method?


FS:[0x124]
ETHREAD
KTHREAD
KAPC STATE
0x44
*EPROCESS
EPROCESS
PCB
(KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB
(KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB
(KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...


FS:[0x124]
ETHREAD
KTHREAD
KAPC STATE
0x44
*EPROCESS
EPROCESS
PCB (KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB (KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...
EPROCESS
PCB (KPROCESS)
*UniqueProcessID
ActiveProcessLinks
LIST_ENTRY {
*FLINK
*BLINK }
...
*Token
ImageFileName[]
PEB
...

Rootkit: Hidden process

Windows thread scheduler's queues:

- KiDispatcherReadyListHead : ready to execute
- KiWaitInListHead
- KiWaitOutListHead

Waiting for an event

Rootkit: Anti-detection methods

- Controling virtual memory access (Translation Lookaside Buffer Hack)
- SSDT Hooks
- Redirecting physical memory access

Rootkit: Neo

Can you trust results you have seen on a possibly infected machine?

RootKit: Anti-anti-detection methods

Cut power / Minimal OS

RootKit: Meh

BIOS-rootkit? ;P

RootKit: Lol

Freezing circuits

References

- http://www.terena.org/activities/tf-csirt/meeting27/oesterberg-rootkits.pdf
- http://www.securelist.com/en/analysis/204792016/Rootkit_evolution
- http://www.dfinews.com/articles/2013/11/understanding-rootkits-using-memory-dump-analysis-rootkit-detection
- http://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf
- https://blogs.mcafee.com/mcafee-labs/targeting-zeroaccess-rootkits-achilles-heel
- http://www.carnal0wnage.com/papers/rootkit_for_the_masses.pdf
- https://www.symantec.com/avcenter/reference/when.malware.meets.rootkits.pdf
- http://blackhatlibrary.net/Azazel
- http://www.blackhat.com/presentations/bh-dc-07/Rutkowska/Presentation/bh-dc-07-Rutkowska-up.pdf
- http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf