Advanced Windows 2000 Rootkits Detection

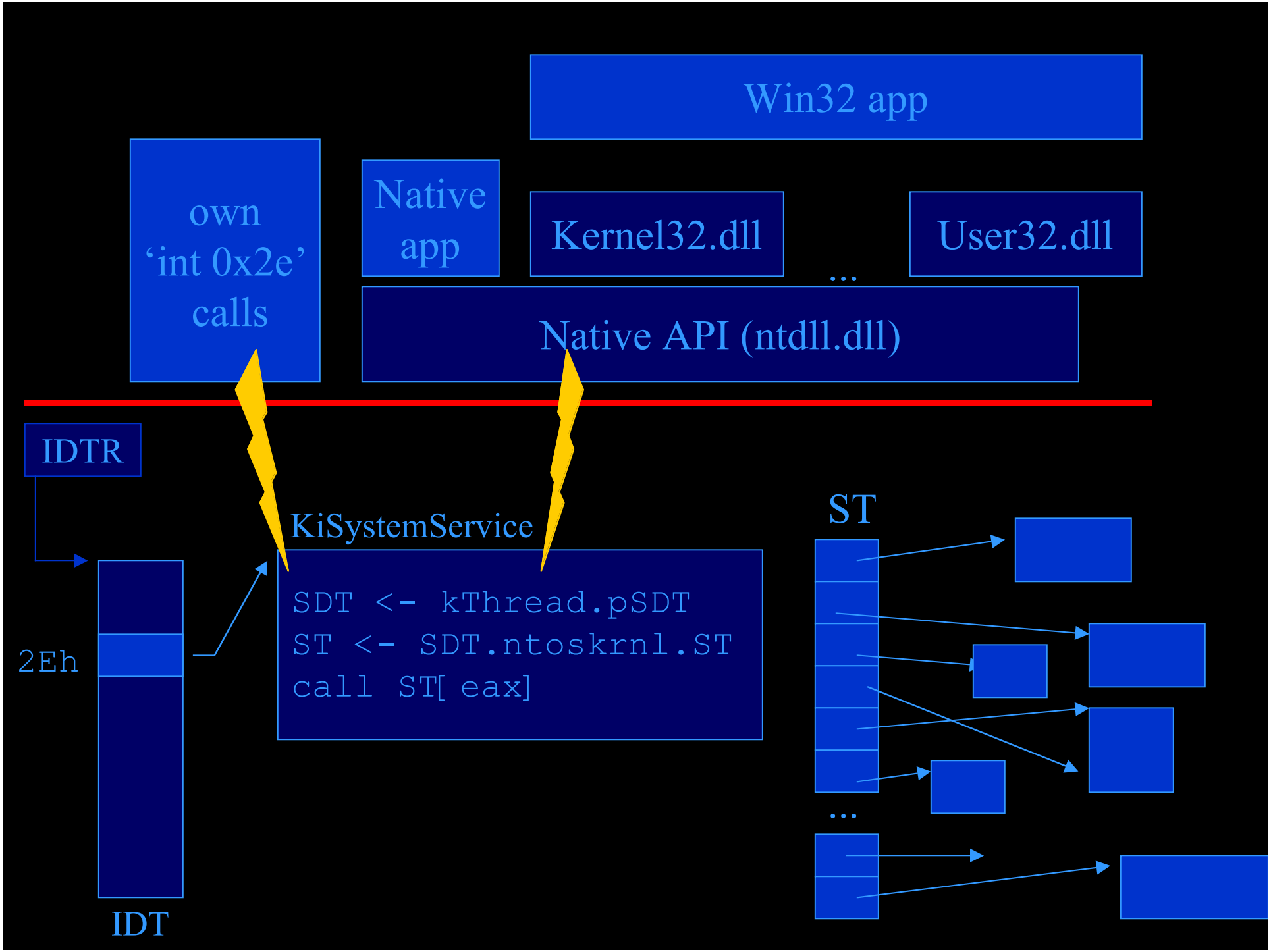
Jan K. Rutkowski

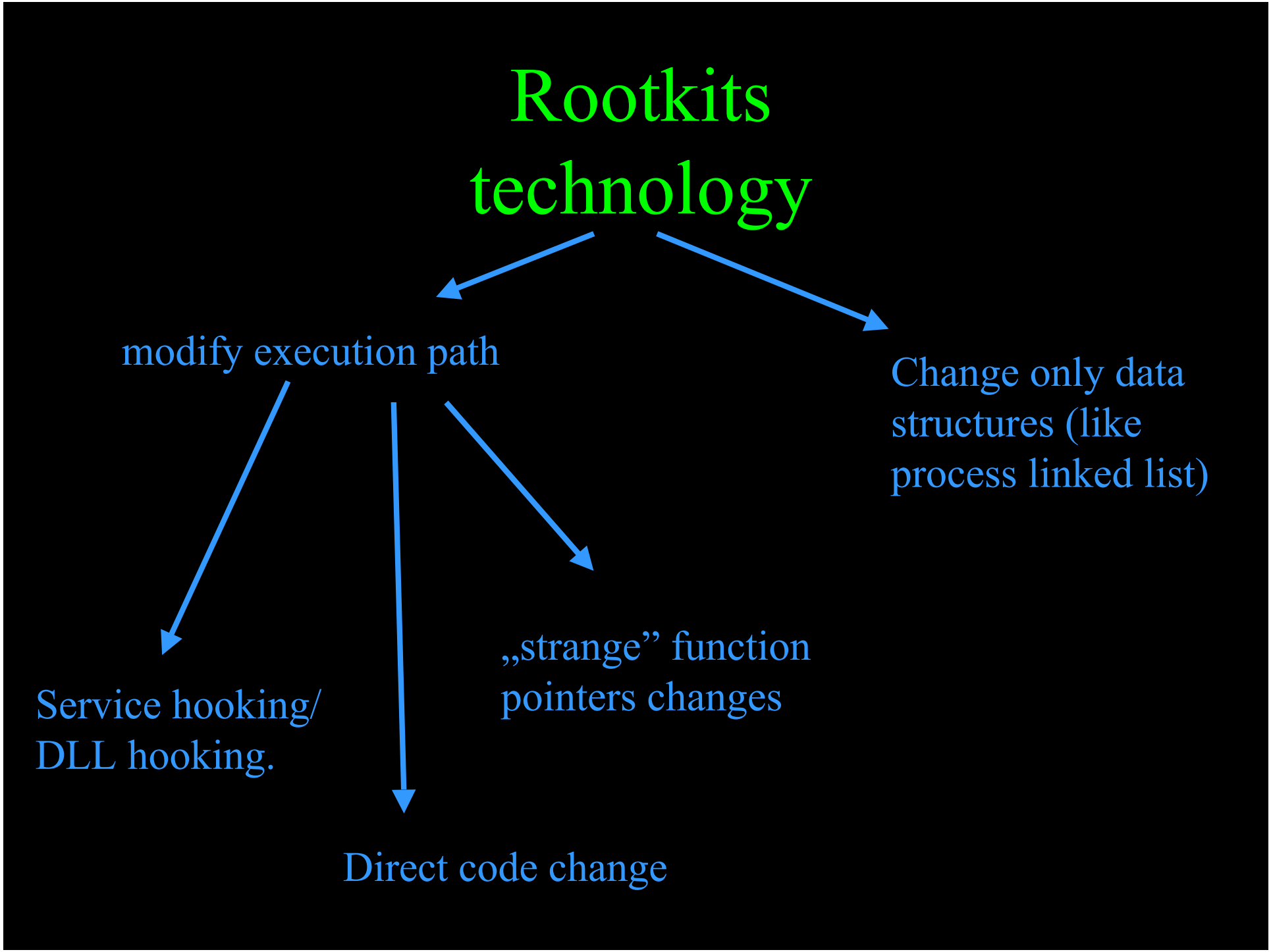
jkrutkowski@elka.pw.edu.pl

- Rootkits overview
- Prevention systems and their vulnerabilities
- Advanced rootkit technology
- Traditional detection
- Execution Path Analysis
  - Idea
  - Implementation
  - Reliability
  - Cheat resistance
- Detection summary

What can rootkit do?

- Hides processes
- Hides files or real file contents
- Hides registry keys or real key values
- Adds backdoor
- Hide backdoor presence
  - from local admin
  - from remote scanning
- Sniffs something
- etc, etc, ...


Win32 app
own
'int 0x2e'
calls
Native
app
Kernel32.dll
...
User32.dll
Native API (ntdll.dll)
IDTR
KiSystemService
SDT <- kThread.pSDT
ST <- SDT.ntoskrnl.ST
call ST[eax]
2Eh
IDT
ST
...


Rootkits technology
modify execution path
Change only data structures (like process linked list)
Service hooking/ DLL hooking.
Direct code change
„strange” function pointers changes

Installing rootkit

- Userland rootkit
  - Changing files on disk (depreciated)
  - Changing memory of other processes (*OpenProcess*, `\Device\PhysicalMemory`)
- Kernel mode rootkit
  - Kernel Driver
    - SCM API (official API to load module)
    - *ZwLoadDriver*
    - *ZwSetSystemInformation* (Greg Hoglund)

    } Appropriate registry key required
  - `\Device\PhysicalMemory` (crazylord)
  - Kernel Overflow (no proof-of-concept yet)

Kernel protection

- Driver Signing
  - Doesn't actually protect against rootkits.
- Integrity Protection Driver
  - from Pedestal Software
  - Open source
- Server Lock
  - From Watchguard
  - Commercial software, about $1000 per server.

Integrity Protection Driver

- Is a kernel driver
- Hooks some system services to forbid loading of any NEW module.
- Standard drivers found in the `\WINNT\System32\Drivers` directory are still allowed to be loaded.
- Activates protection 20 minutes after the module has been loaded.
- Reboot is needed to remove the IPD after that time

IPD hooks

- ✓ ZwOpenKey/ZwCreateKey/ZwSetValueKey (protects `\HKLM\System\CurrentControlSet\Services`)
- ✓ ZwOpenSection (block `\Device\PhysicalMemory`)
- ✓ ZwCreateFile/ZwOpenFile (block `\Device\Harddisk*`, etc...)
- ✓ ZwCreatLinkObject (to prevent cheating *ZwOpenSection* and *Zw{Create,Open}File*)
- ✓ ZwSetSystemInformation:
  - ✓SystemLoadAndCallImage
  - ✓SystemLoadImage
- ✓ ZwOpenProcess prevent Runtime Process Infection.

IPD: Bug history

- ZwSetSystemInformation not hooked (Hoglund, 2000),
- Bypass of `\Device\PhysicalMemory` protection (crazylord, 2002),
- Bad logic in *restirctEnabled()* (2002),
- Drivers directory protection bypass:
  - with ‘subst’ (2002),
  - Problem with driver’s without *ImagePath* field (2003),
- Raw disk access and driver file replacement (2003),
- More problems with ZwSymbolicLinkObjects (2003).

IPD: problems with *CreateSymbolicLinkObject()*

```
C:\spool>funWithLinks.exe
creating link: \hak1 --> \Device
creating link: \hak2 --> \Device\PhysicalMemory [failed]
creating link: \hak3 --> \
creating link: \hak4 -->  [failed]
creating link: \Device\hak5 --> \Device
creating link: \Device\hak6 --> \??\GLOBALROOT

trying to open for READ|WRITE:
opening \Device\PhysicalMemory ... [failed]
opening \hak1\PhysicalMemory ... [it worked!]
opening \hak2 ... [failed]
opening \hak3\Device\PhysicalMemory ... [failed]
opening \Device\hak4\PhysicalMemory ... [failed]
opening \Device\hak5\PhysicalMemory ... [it worked!]
opening \Device\hak5\hak5\PhysicalMemory ... [it worked!]
opening \??\GLOBALROOT\Device\PhysicalMemory ... [it worked!]
opening \Device\hak6\hak1\PhysicalMemory ... [it worked!]
```

IPD fixes

- The last version of IPD blocks *ZwCreateSymbolicLinkObject()* totally;)
- IPD shows that it is very difficult to write good protection program for third party company (i.e. not OS vendor)

ServerLock

- Consists of a driver module and nice GUI configuration program.
- Similar idea to IPD – do not allow any new module to be loaded, it hooks:
  - Registry key manipulation functions,
  - ZwSetSystemInformation,
  - Protects \Device\PhysicalMemory
- Possibility to also protect files from changes (by viruses for e.g.)

Problems with ServerLock

- ZwSetSystemInformation allows ‘trusted’ processes to load driver. Trusted process is considered one, that was created from program file, which is protects against changes. This can be abused by DLL injection for e.g. (2003).
- This has been fixed in the new version (2003).
- Vendor refuse to provide details when calling ZwSetSystemInformation is allowed.

Problems with ServerLock cont.

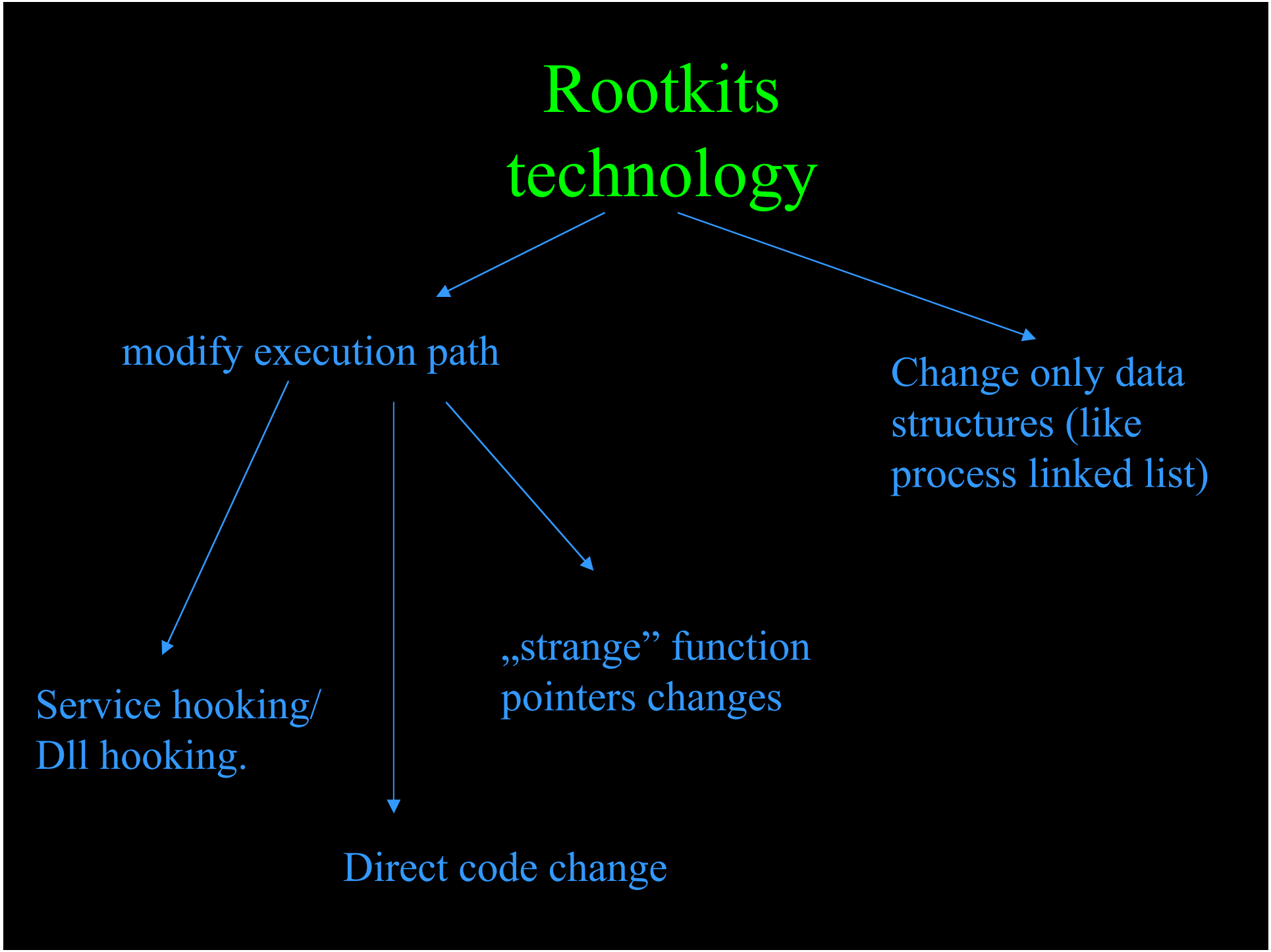
- Doesn't hook *ZwOpenSection*, so RPI is possible. Rootkits like *hxdef* can be installed.
- Similar problems with accessing `\Device\PhysicalMemory` through symlinks.
- This has been reported to Watchguard at the beginning of 2003. These issues has not been repaired yet...

Kernel Overflows

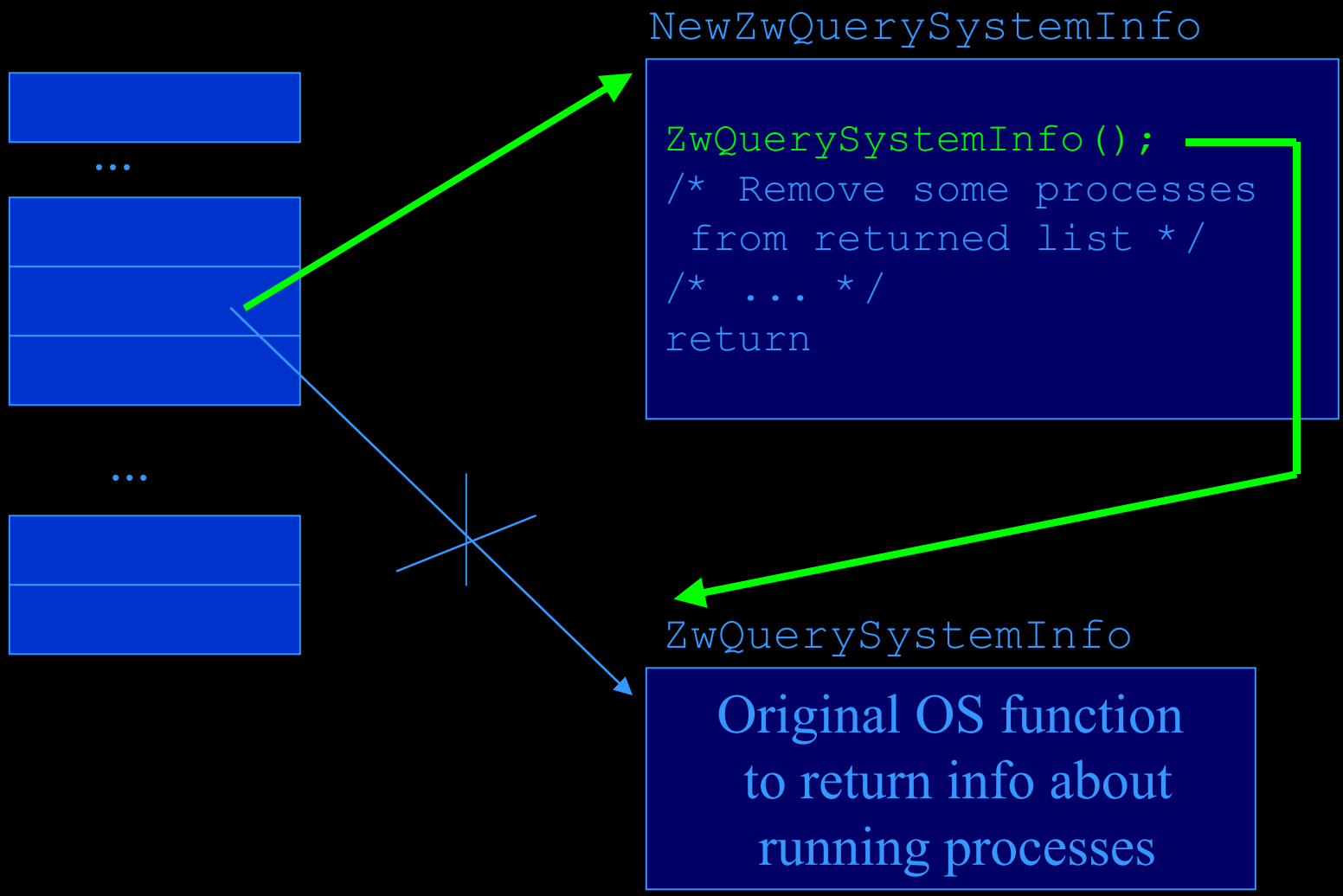
- Attacker can find a bug in one of many kernel drivers and get into the kernel.

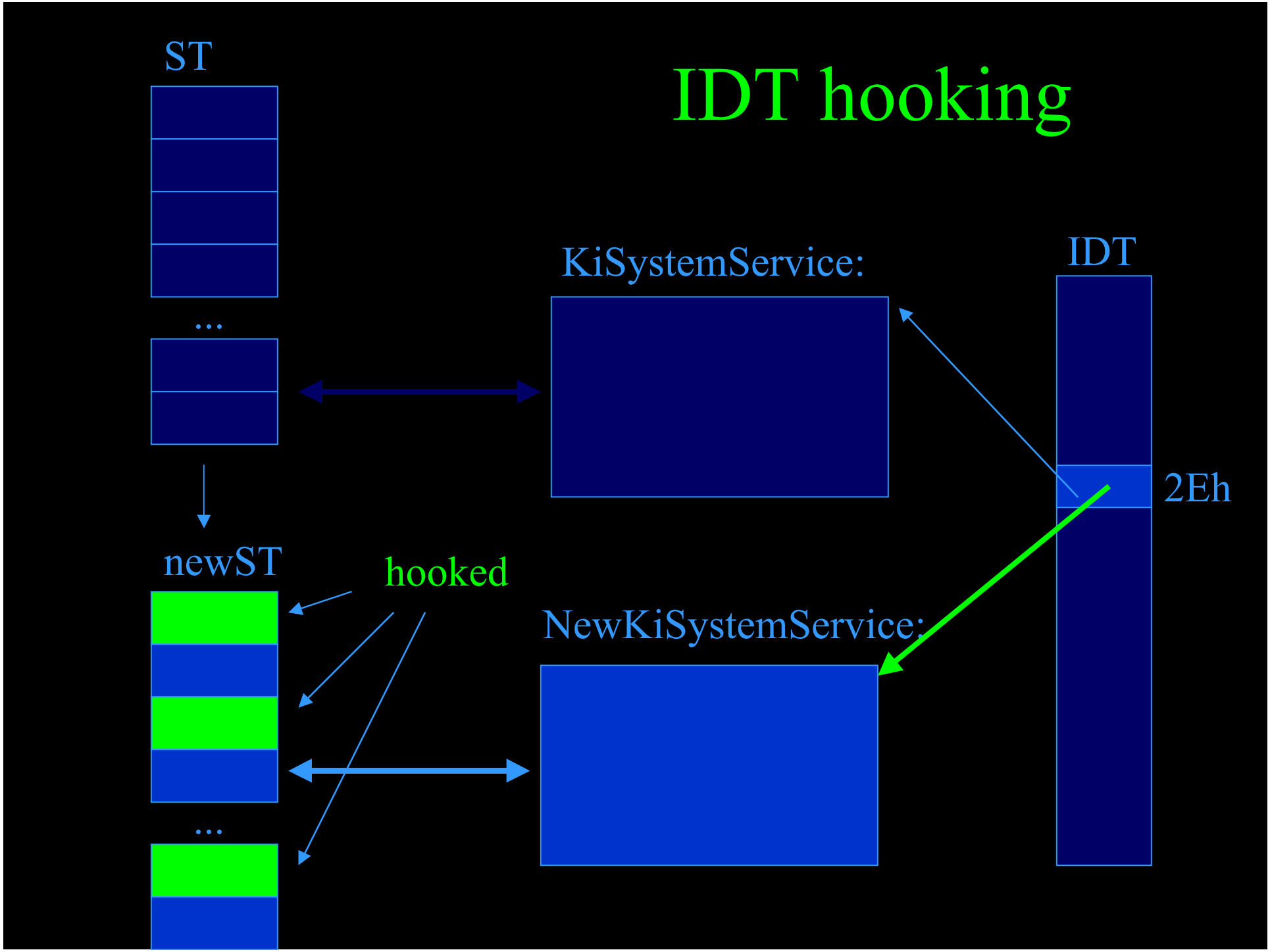
Protection of Windows kernel

- Although never gives 100% (vide kernel overflows) is a very good idea.
- Something similar to *securelevel* from *BSD, should be implemented on Windows. Probably it would be best done by Microsoft.
- We see however, that we cannot relay fully on prevention, so lets discuss detection...

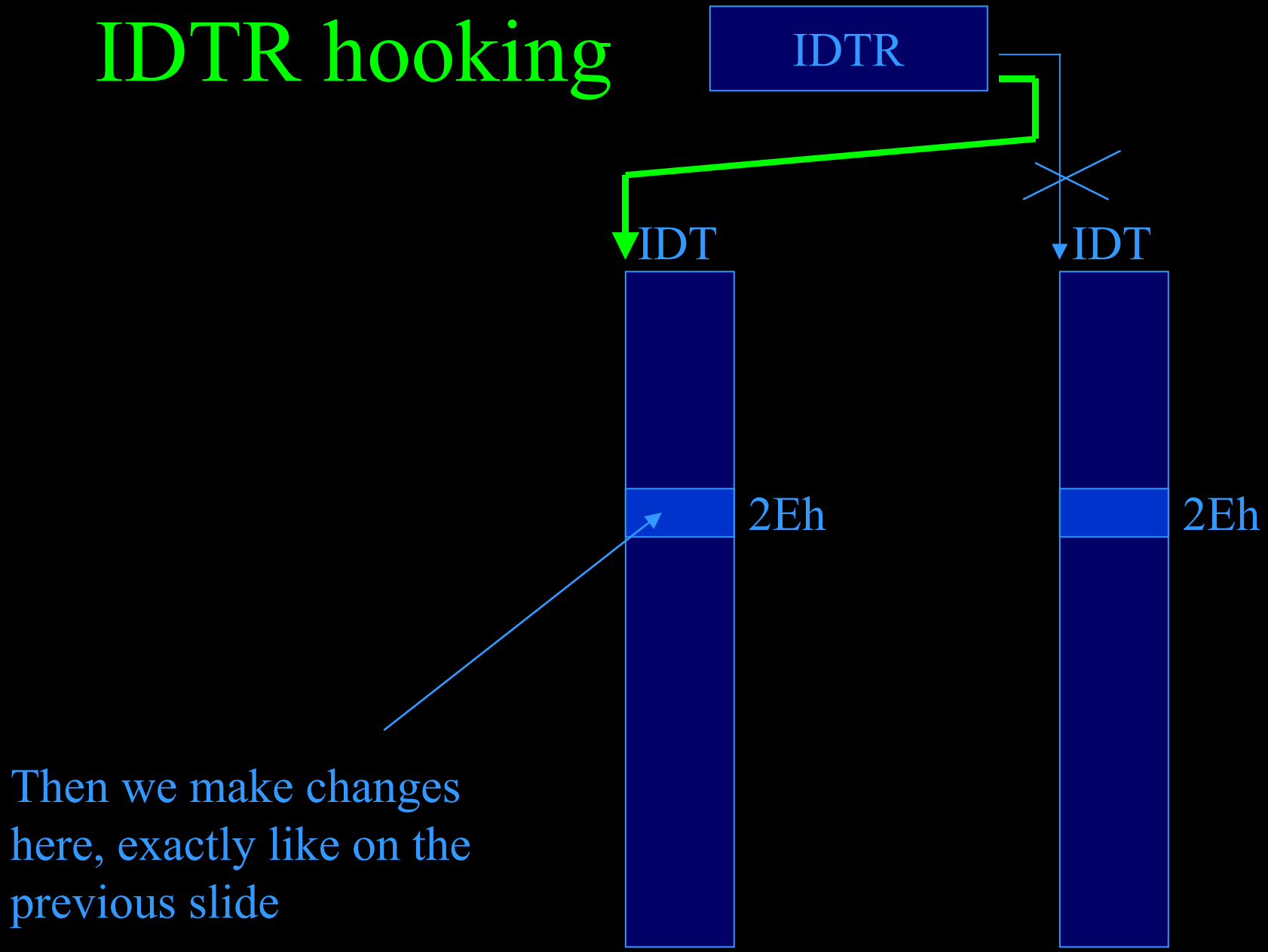

Rootkits technology
modify execution path
Change only data structures (like process linked list)
Service hooking/ Dll hooking.
„strange” function pointers changes
Direct code change

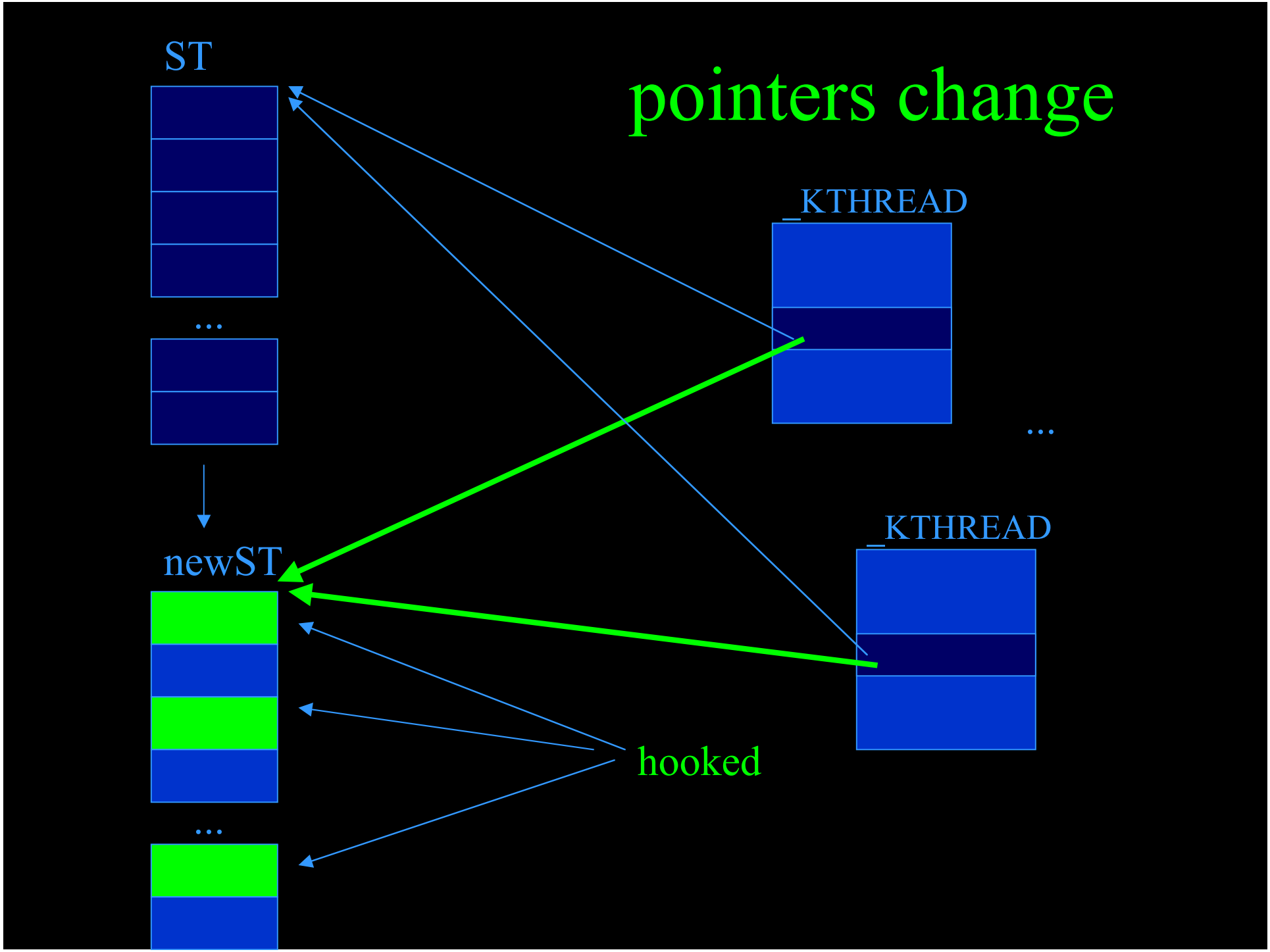
Classic ST hooking



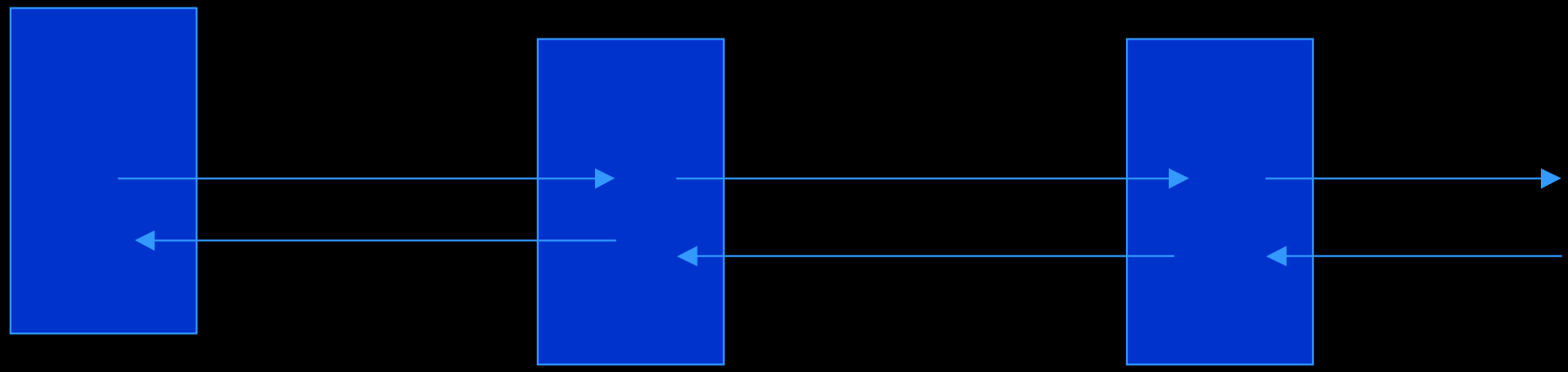

ST
IDT hooking
KiSystemService:
IDT
...
2Eh
newST
hooked
NewKiSystemService:
...

IDTR hooking




ST
pointers change
_KTHREAD
...
...
_KTHREAD
newST
hooked
...

Process are linked in double list

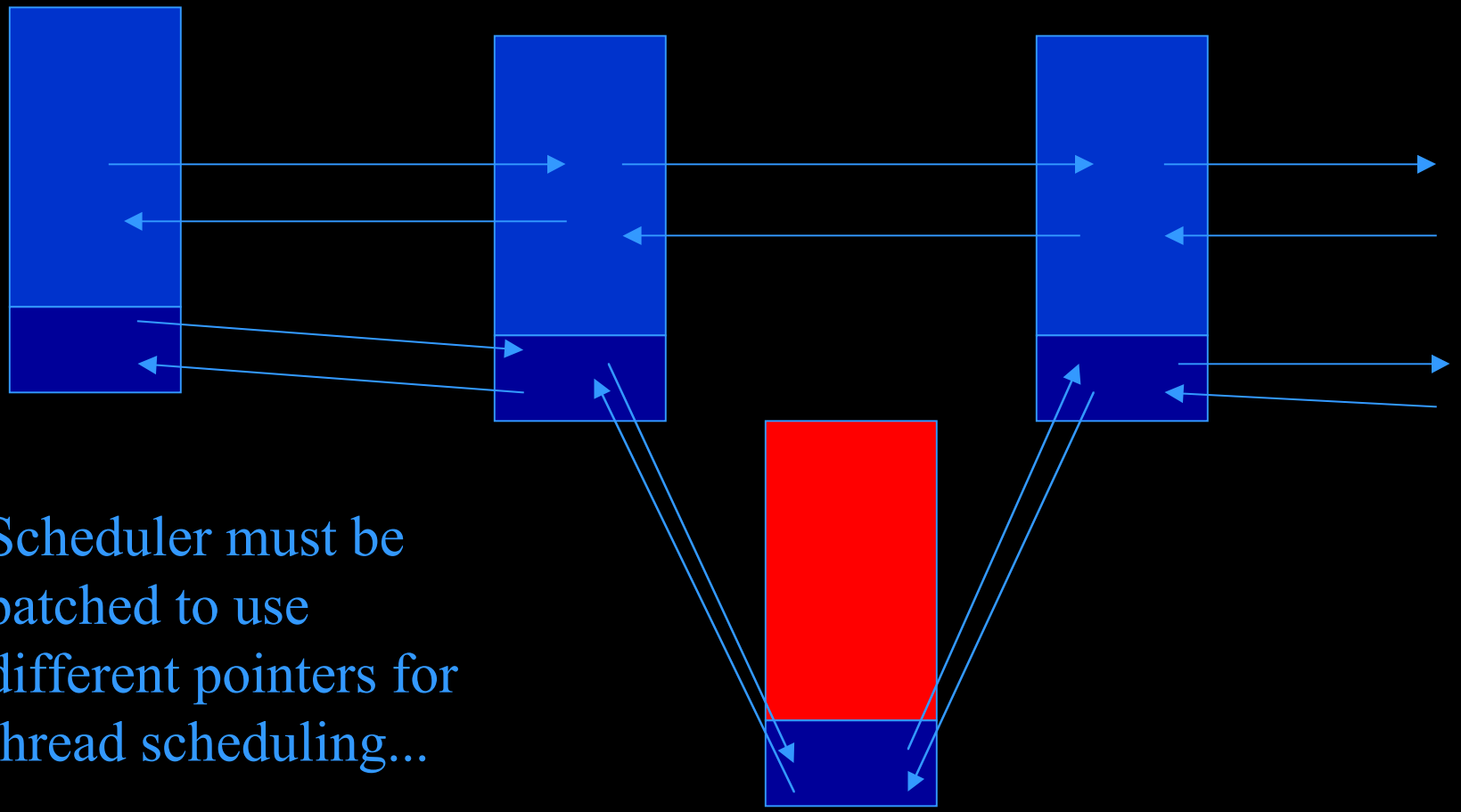


- _EPROCESS.ActivePorcessLink (*fu* rootkit)
- _KPROCESS.ReadyListHead
- KiDispatcherReadyListHead – queues of ready thread
- other queues used by Dispatcher

fu rootkit (by fuzen_op)

- _EPROCESS.ActiveProcessLinks
- Filed not used by scheduler
- We can simply unlink process object.
- However, threads from this process must be on some other lists (like KiDispatcherReadyListHead), to obtain some CPU quantum form scheduler...
- We can scan this lists then and unhide hidden processes (threads).

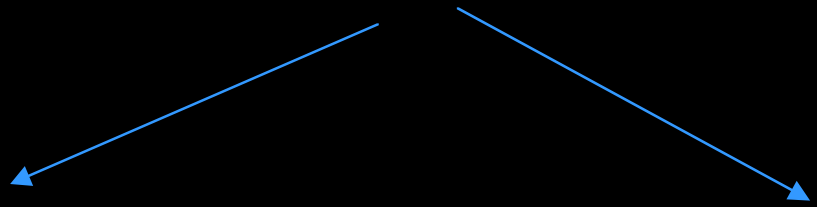
„Shadow” threads list



Scheduler must be patched to use different pointers for thread scheduling...

So, how to detect if our kernel has been compromised or not?

detection



We have some clear system state to compare with (like dump of ST).

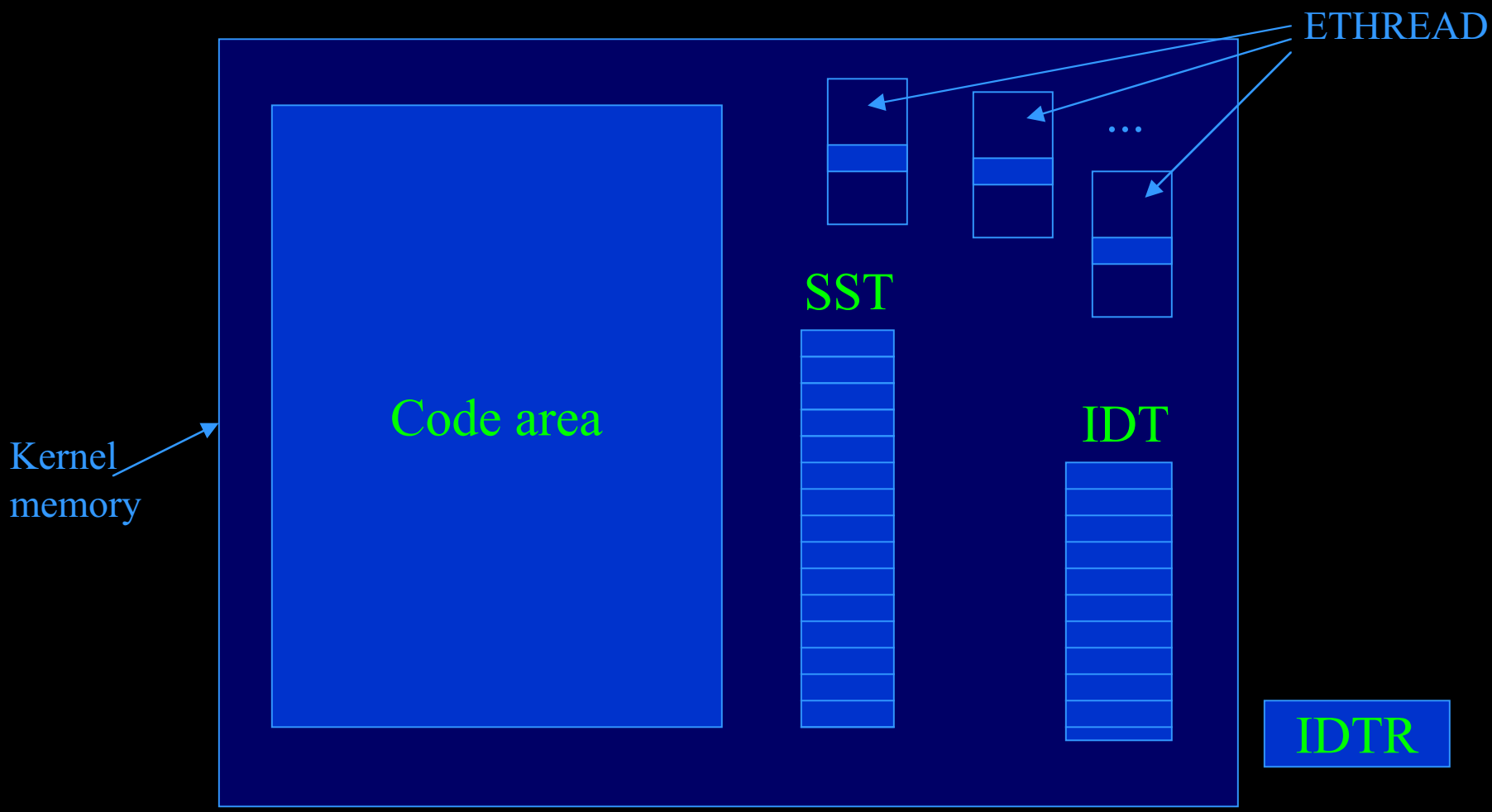
We can analyze only current system, have no info about the system when was clear.

In most cases, we should fit into 1st category.

Comparing areas of kernel memory

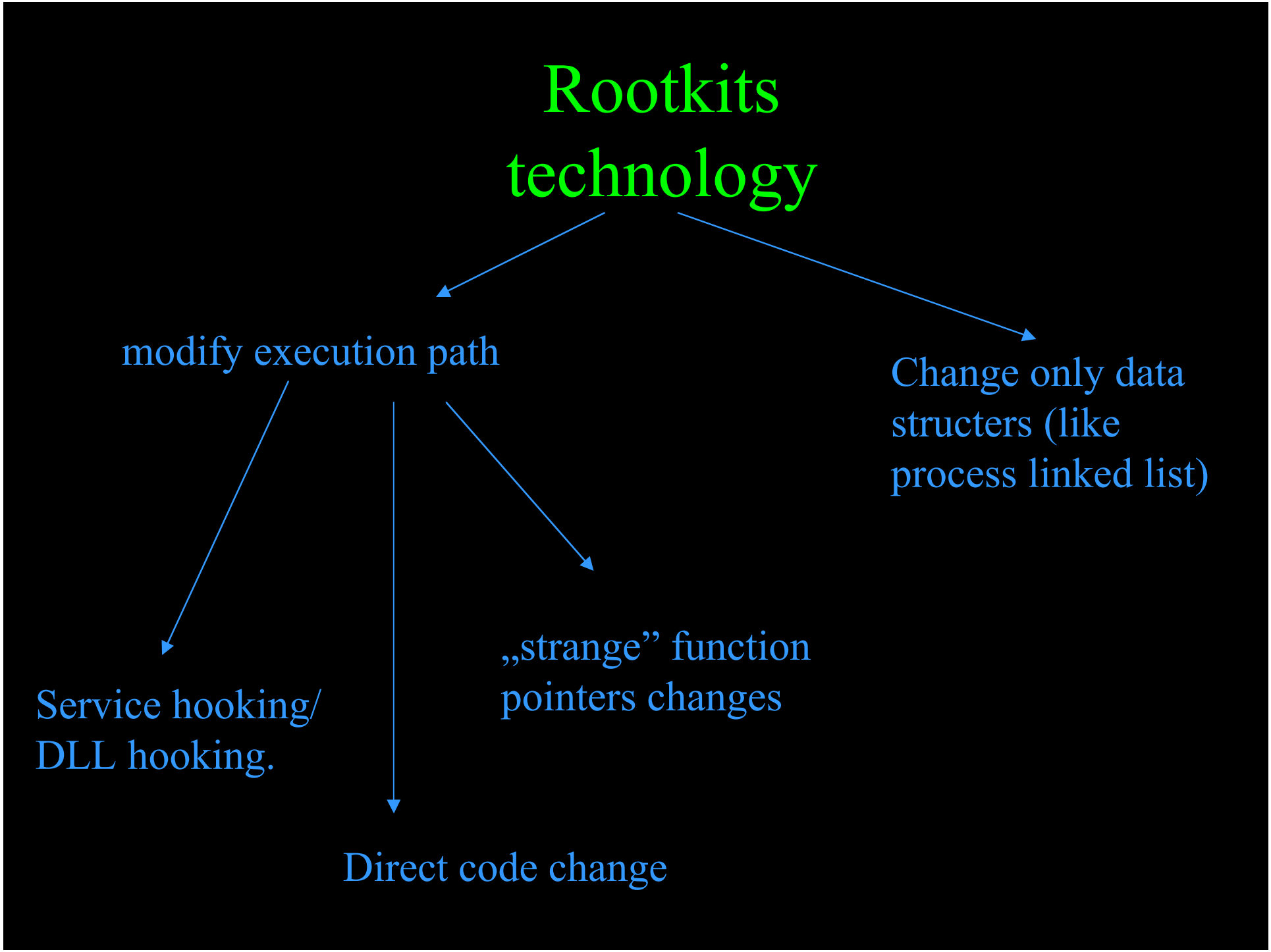
- Make a copy of some kernel memory fragments (like ST, IDT, code) when system is clear (i.e. just after the installation)
- Regularly compare saved contents of memory with the current one.
- Kernel memory should be accessed by means of a kernel driver, not `\Device\PhysicalMemory`.

What memory area should be monitored?



Is it enough?

- Nobody knows...


Rootkits technology
modify execution path
Change only data structers (like process linked list)
Service hooking/ DLL hooking.
„strange” function pointers changes
Direct code change

Reading kernel internal data structures

- When trying to for e.g. find all processes we can use internal lists used by scheduler.
- Possible to cheat, by modifying scheduler code to use copy of the original structures. Original structure is then untouched (see „shadow” threads list concept).
- Accessing kernel data structures should be implemented by kernel driver, not through `\Device\PhysicalMemory`.

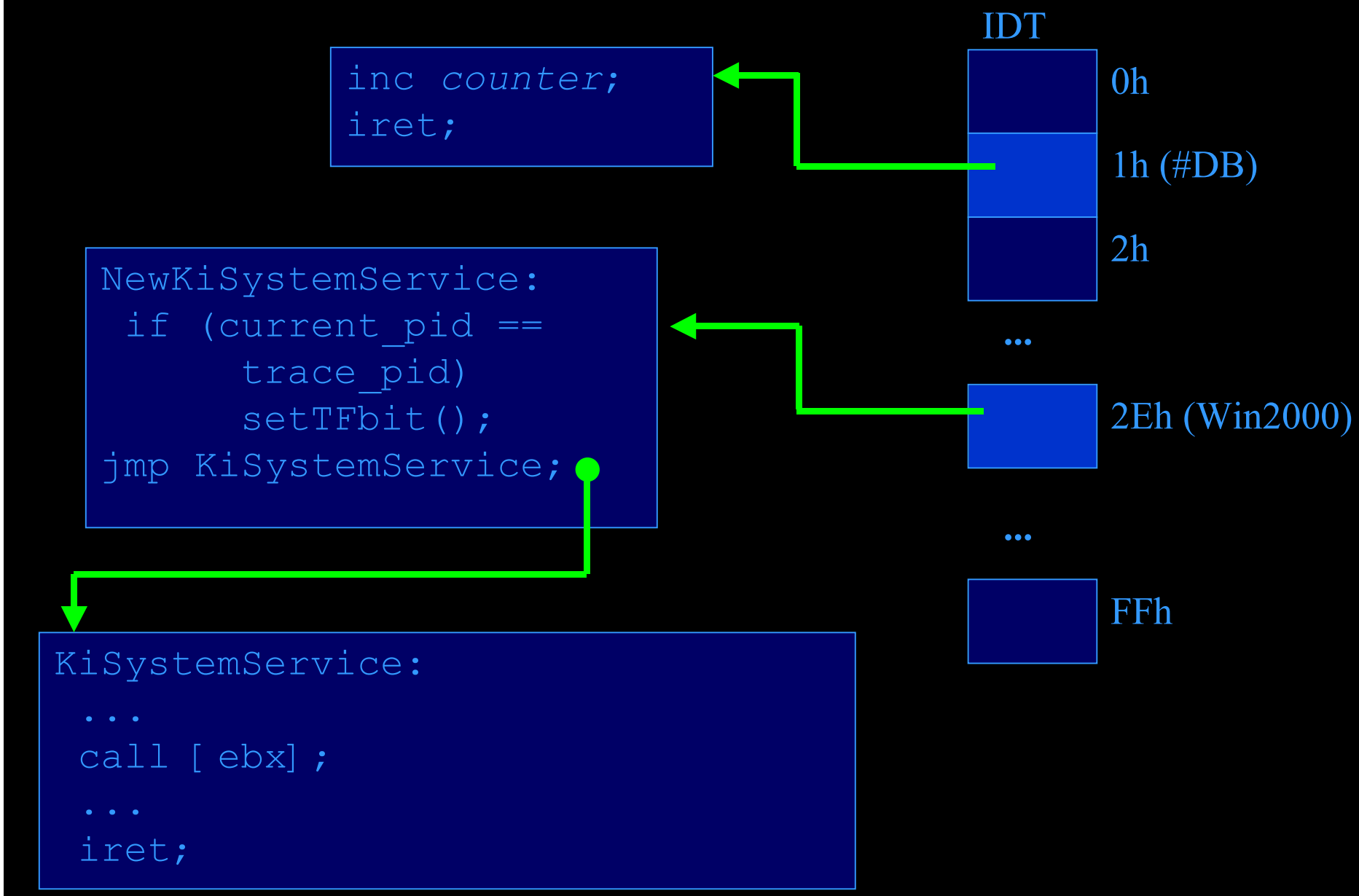
EPA concept

- Measure the number of instructions, which has been executed during some system services

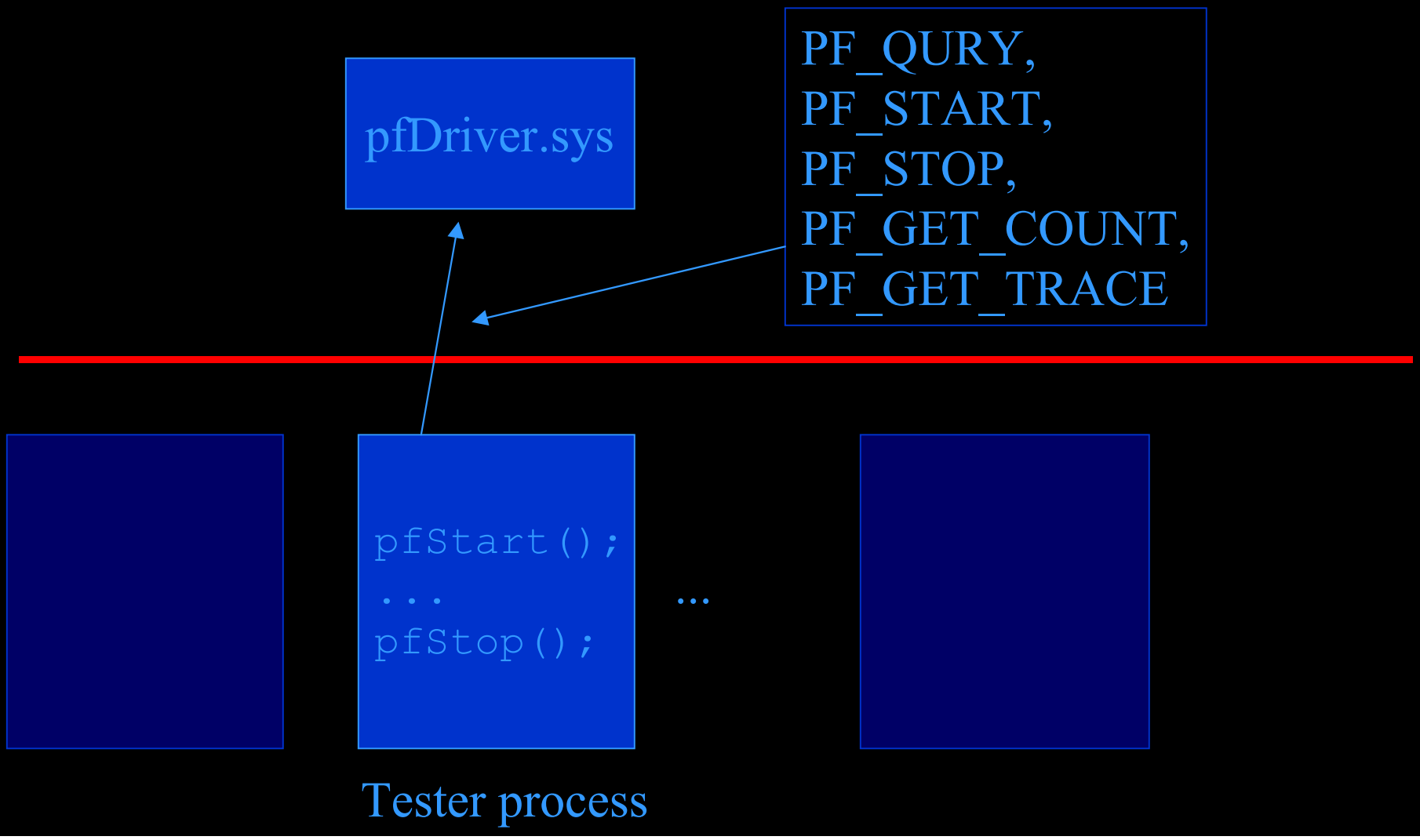
Step mode on IA-32

- Set TF bit in EFLAGS register
- When in step mode, CPU generates #DB exception (Trap class) after the execution of every instruction
- #DB exception handler is stored at IDT[1].
- TF bit is cleared when `int x` instruction is used to enter the kernel mode.

EPA: IDT hooks



Tester process & kernel driver

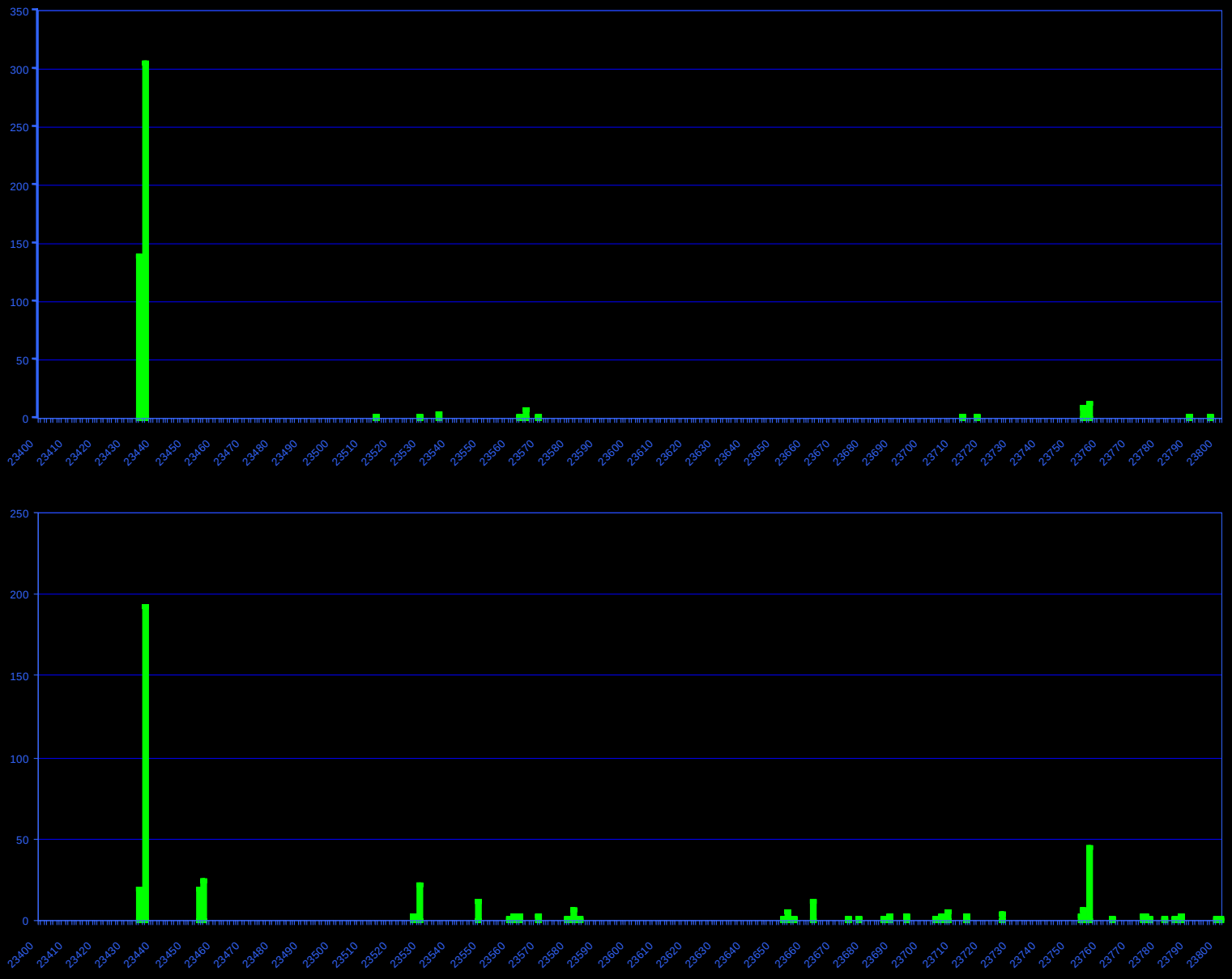


Test example

```
for (int i = 0; i < N; i++) {
      pfStart();
            hFind = FindFirstFile(
                  "C:\\WINNT\\system32\\drivers",
                  &FindFileData);
      pfStop();

      FindClose(hFind);
}
```

We will get *N* samples of test, then we can make a histogram...

FindFirstFile example



Execution Path Recording

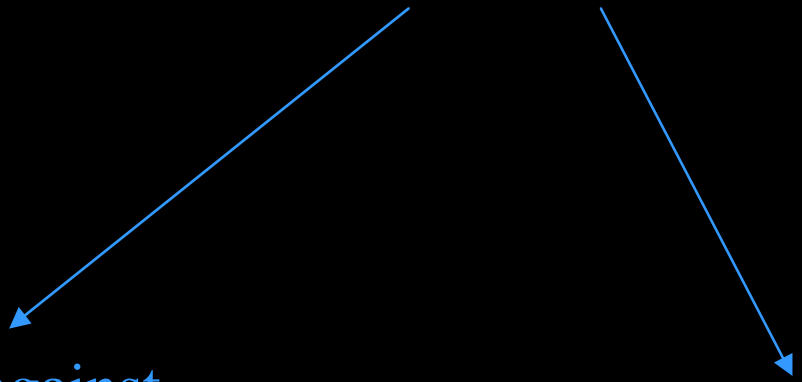
- Sometimes peek's position changes a little (typically less then 20 instructions)
- Is it rootkit or just false positive?
- EPR: see exactly what instructions caused the difference!
- EPR requires deep technical knowledge from user.
- Possibly 'diff –c' can be replaced by some more sophisticated program.

Comparison of two traces:

```
*** RegEnumKey-clear.trace      Sun Jun 29 03:49:21 2003
--- RegEnumKey-current.trace    Sun Jun 29 03:49:21 2003
***************
*** 273,278 ****
--- 273,281 ----
  0x80416f60
  0x80416f63
  0x80416f65
+ 0x80416f67
+ 0x80416f6a
+ 0x80416f6d
  0x80416f74
  0x80416f76
  0x80416f77
```

Extra instructions, which caused false positive.

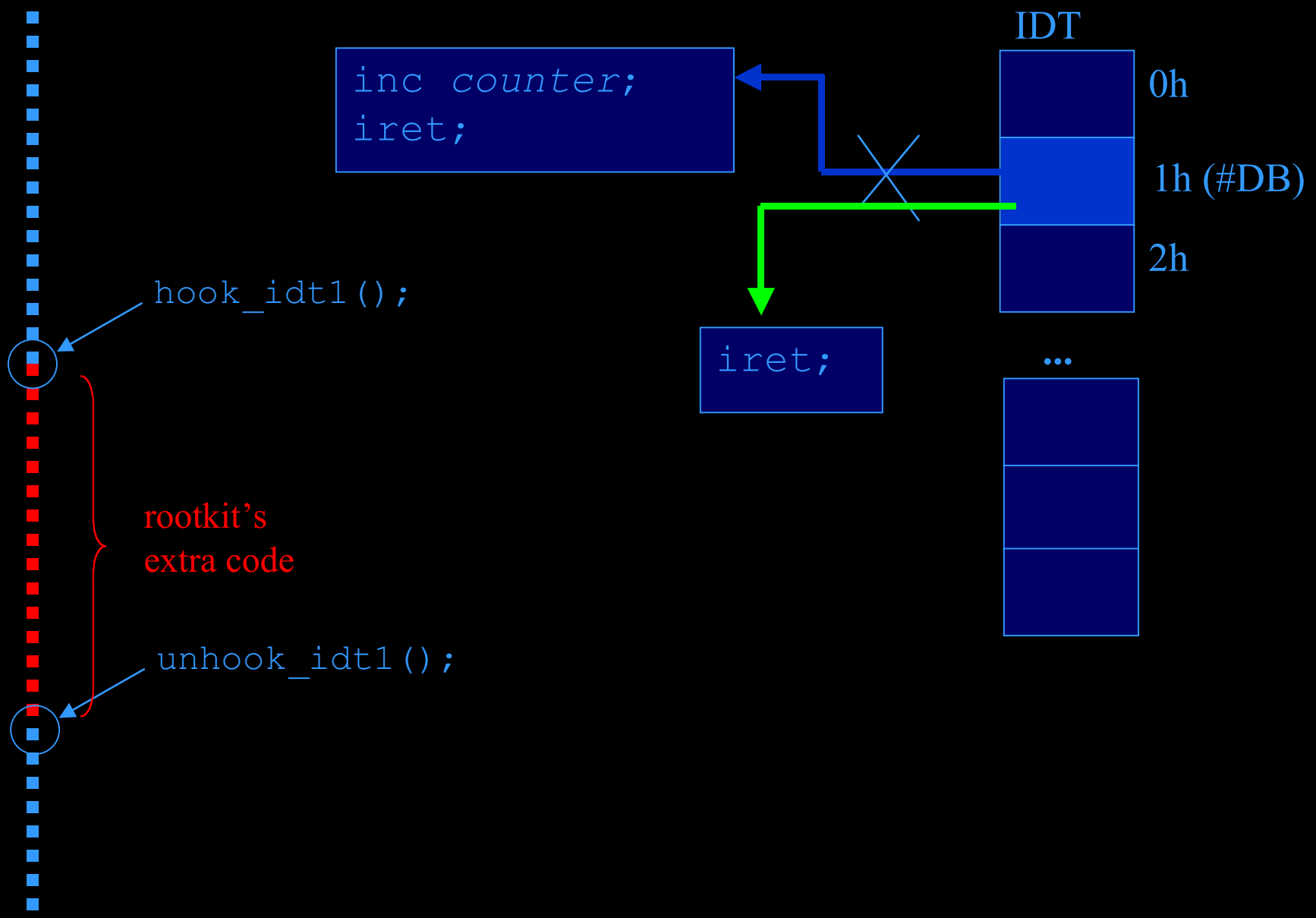
Attacks against EPA



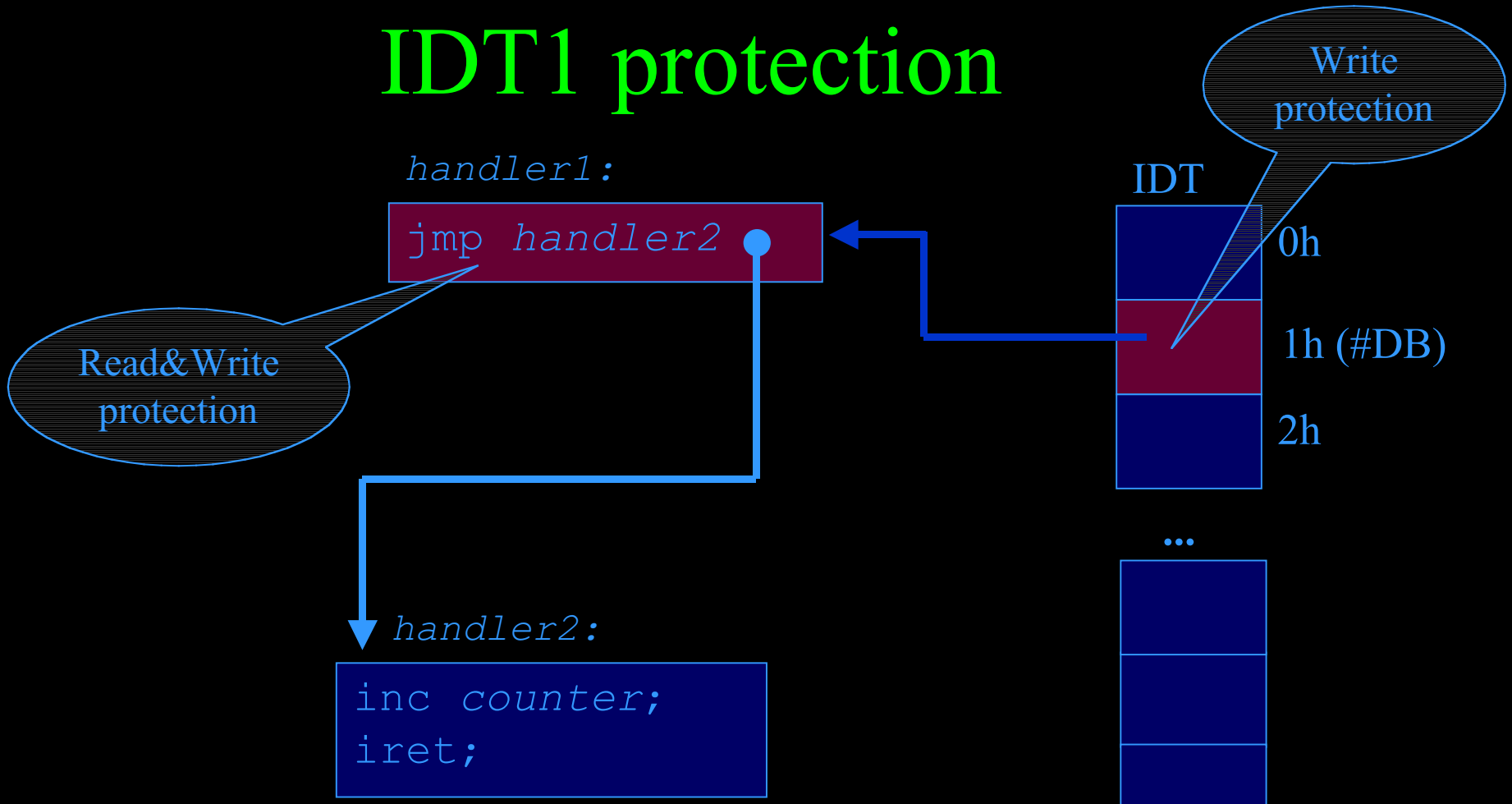
Attacks against specific tool, specific version, specific binary.

More general attacks against EPA concept.

Attack1: IDT1 hook

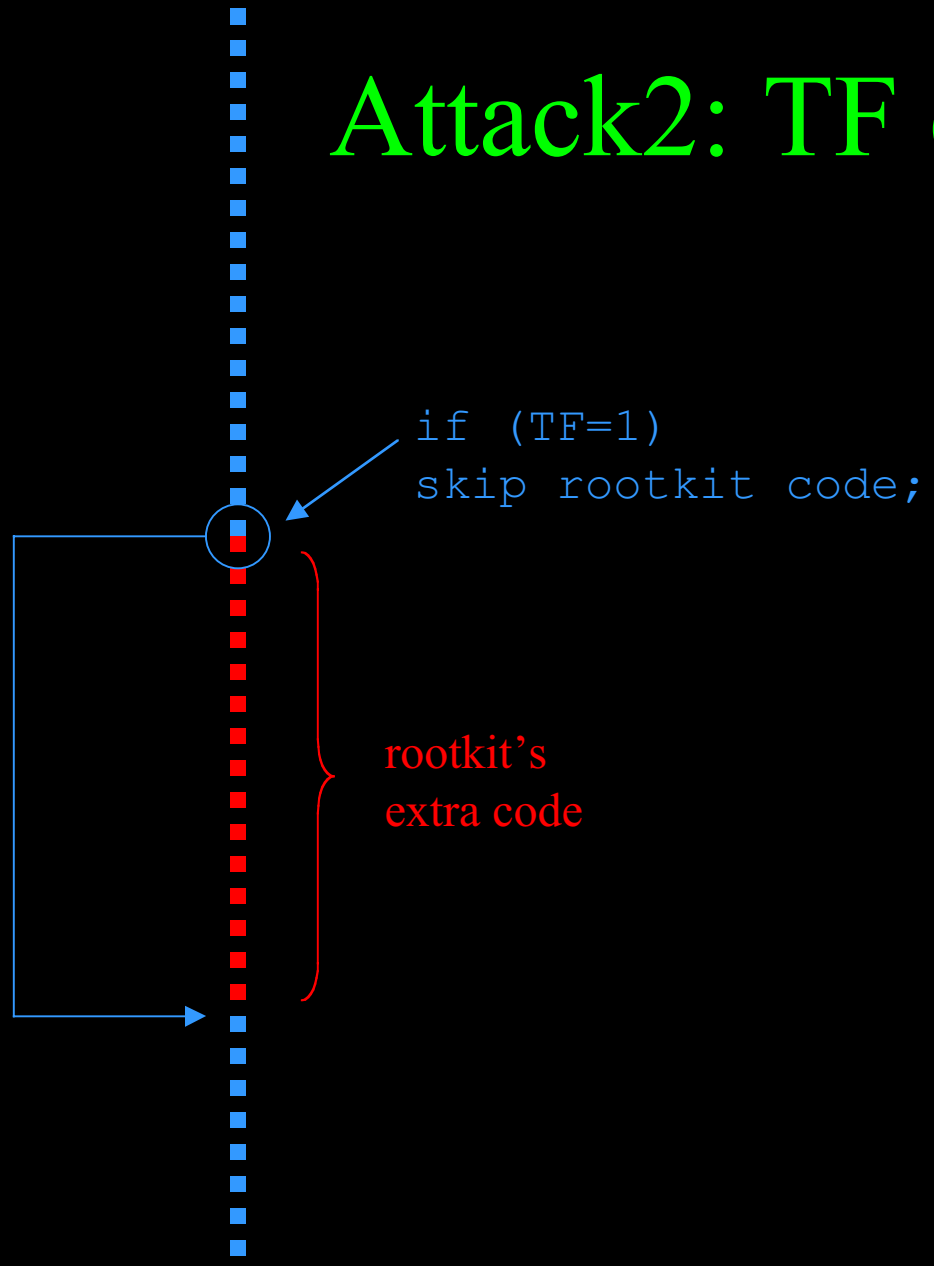


IDT1 protection

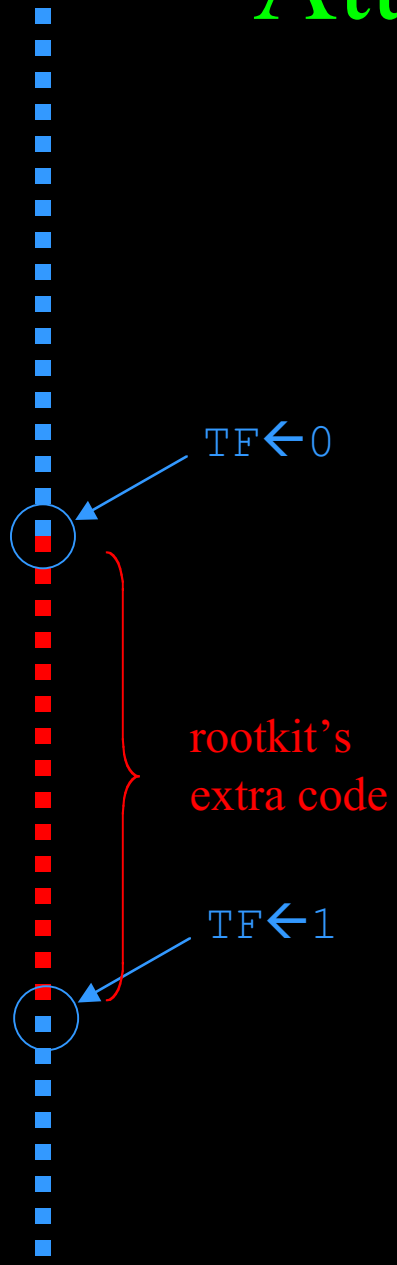


(real *handler2* is much more complicated)

Attack2: TF check

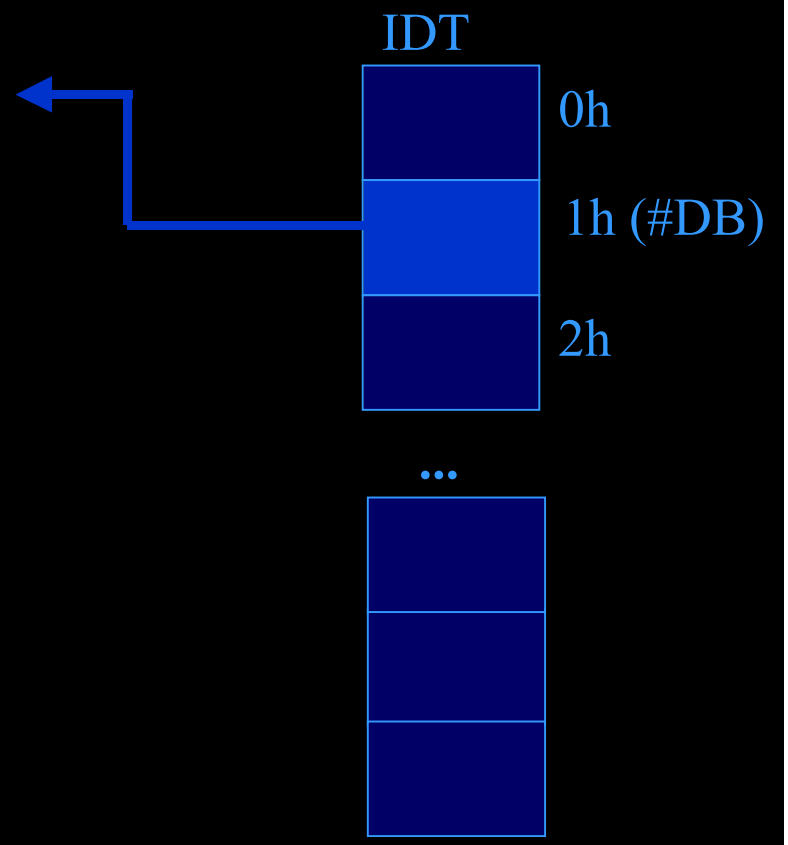


Attack3: disable step mode



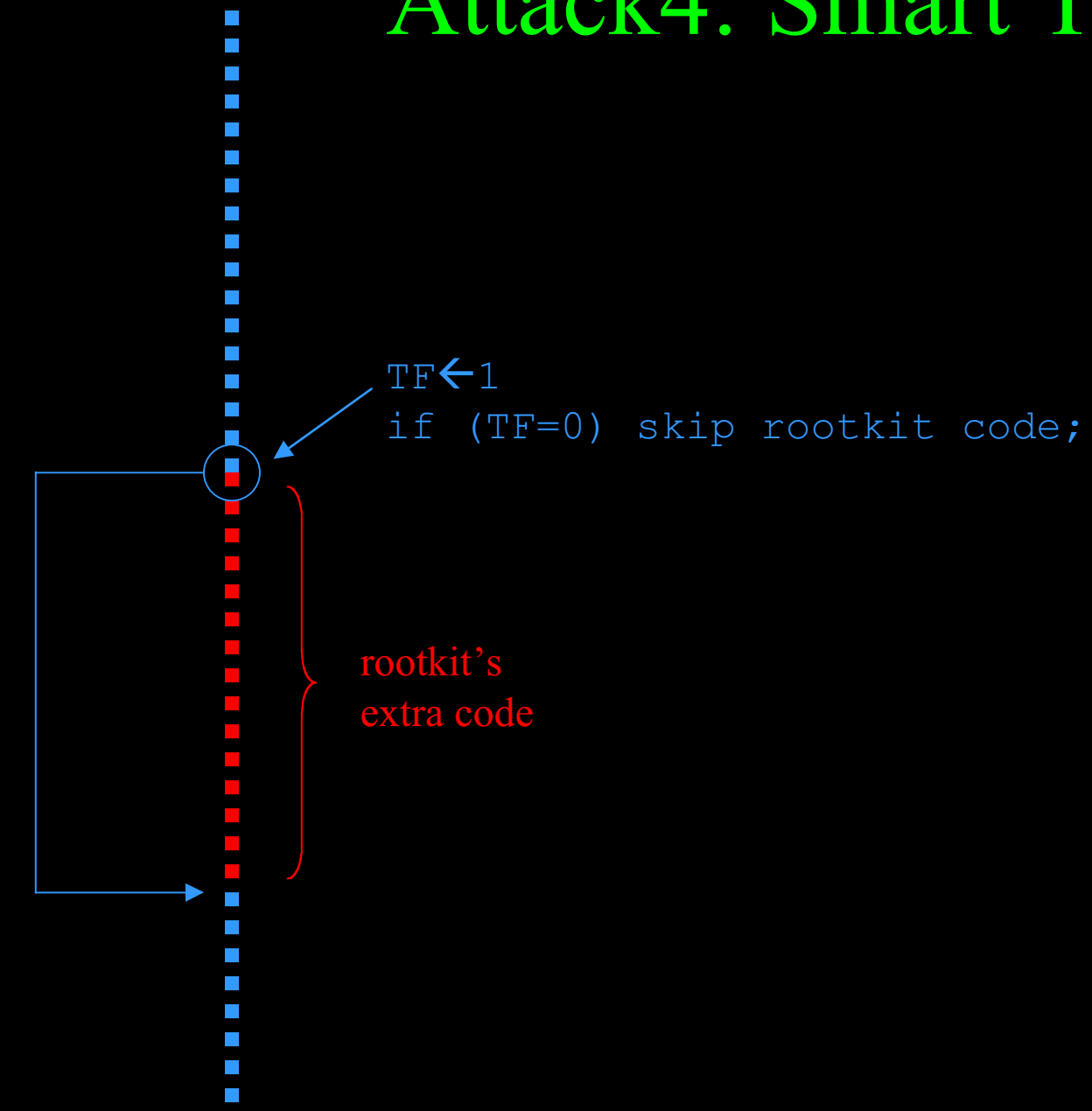
Attack 2&3 defense

```
if (nextInstr == 'popf') {
 setTFbit in [esp]
}

if (prevInstr == 'pushf') {
 clearTFbit in [esp]
}

inc counter;
iret;
```



Note: there are a few instructions similar to popf/pushf, which can access EFLAGS register. They are specified in IA32 manual.

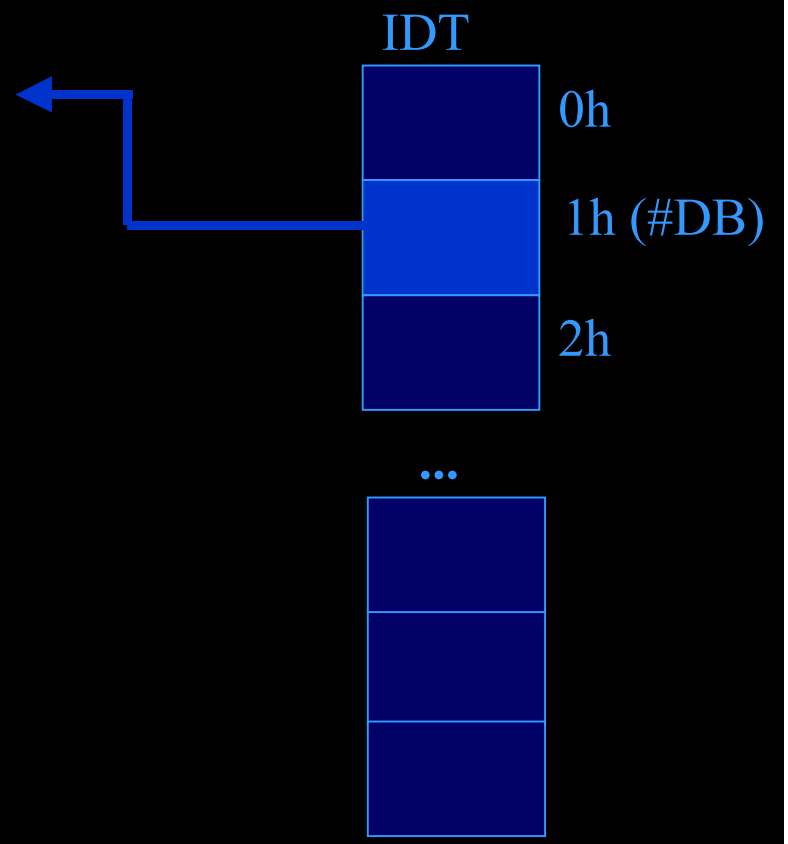
Attack4: Smart TF check



Attack 2,3,4 defense

```
Init: tfbit ← 0;
```

```
if (nextInstr == 'popf') {
 tfbit = getTFbit from [esp];
 setTFbit in [esp];
}

if (prevInstr == 'pushf') {
  setTFbit in [esp] to tfbit;
}
inc counter;
iret;
```



Note: there are a few instructions similar to popf/pushf, which can access EFLAGS register. They are specified in IA32 manual.

Attacks against specific program

- Hard to defend
- Hard to rootkit author, when more then one tool, more then one version exists.
- Defense: polymorphic code generation for every machine (during installation phase).

Practical Detection Toolkit

...should combine:

- File & Registry integrity checker
- Kernel memory integrity checker
  - Code
  - IDT, IDTR
  - ST, pointers to ST
- kernel structures reader
  - Processes/Threads lists
  - ?
- EPA