EKO-PARTY 2011

```
DDDD   EEEEE EEEEE PPPP    BBBB  OOOOO OOOOO TTTTT
D   D  E     E     P   P   B   B O   O O   O   T
D   D  EEE   EEE   PPPP    BBBB  O   O O   O   T
D   D  E     E     P       B   B O   O O   O   T
DDDD   EEEEE EEEEE P       BBBB  OOOOO OOOOO   T
```

Nicolás A. Economou

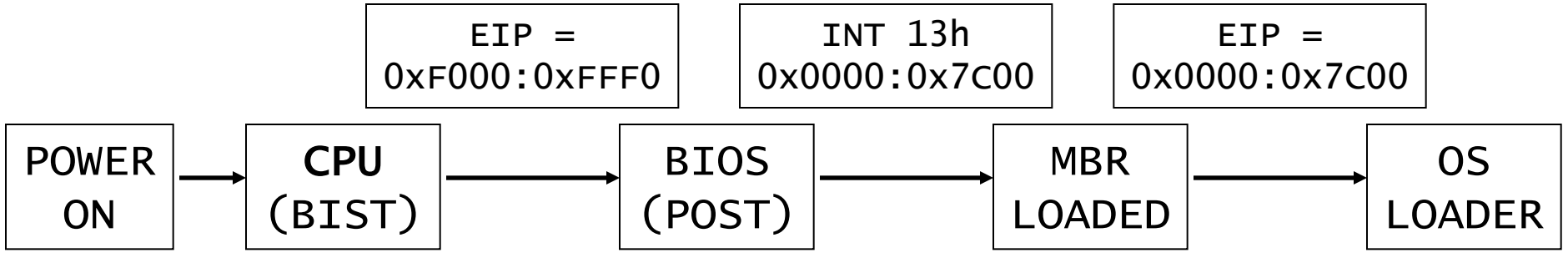
Andrés Lopez Luksenberg

INTRO

- ***"There have been as many new MBR threats found in the first seven months of 2011 as there were in previous three years ...".***

  Symantec Intelligence Report: August 2011

Boot Mechanism



Boot Mechanism

- Luego, se ejecuta el codigo que levanta el proximo stage del LOADER del OS

- Ej. Windows 7
  - 1. MBR
  - 2. Bootmgr (file)
  - 3. Winload.exe (embebido en el bootmgr)
  - 4. Winload.exe (file)
  - 5. Resto de los files (kernel, drivers, etc)

Ejecuting from the beginning

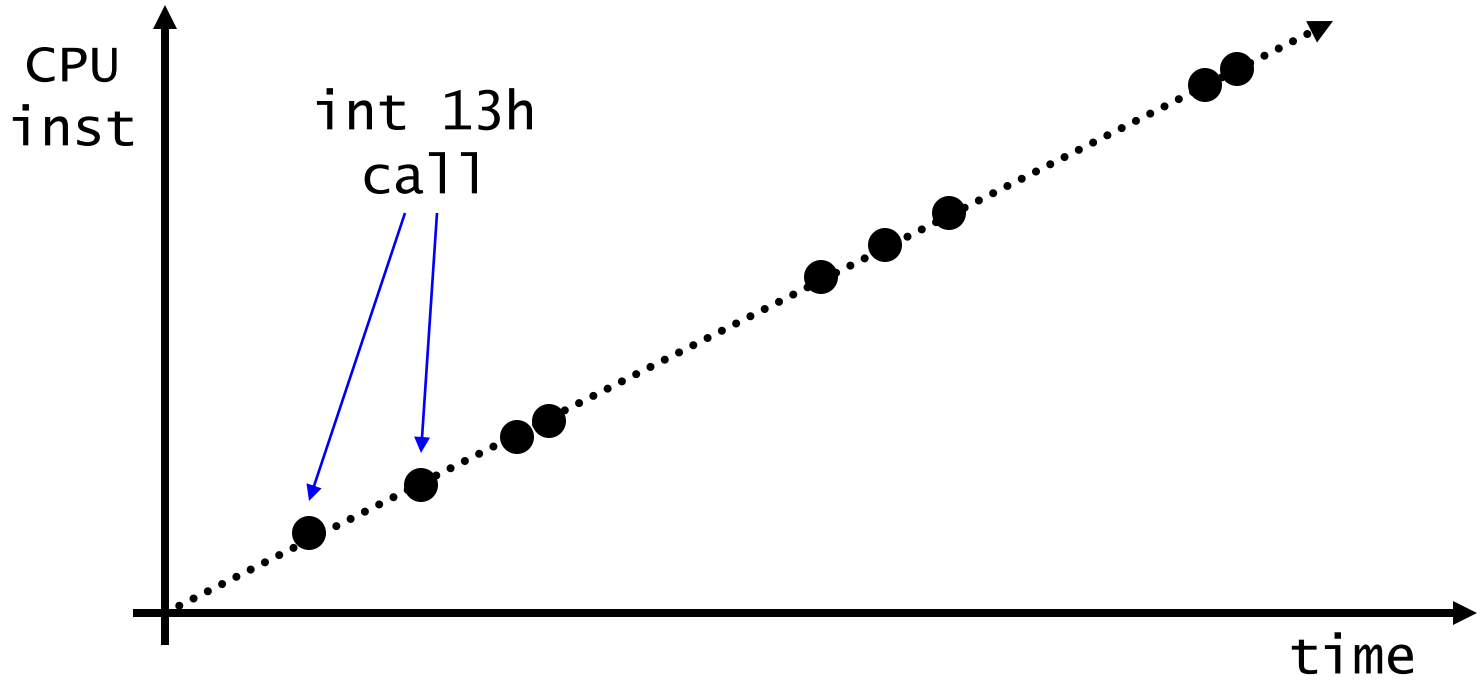
- Estando en el MBR (Disco Rigido)
- Booteando desde un dispositivo removible (CDs, PENDRIVEs, ZIPs, FLOPPY)
- PXE (Network)
- BIOS modificado (ejecutando antes de 7C00h).
- WARM BOOT (el segundo principio)

Common Ways (File Patching)

- Usando la “INT 13H” del BIOS
  - Pattern Matching:
    - » Recorrer el disco y patchear
    - » Hookear mientras el OS se carga y patchear (**StonedBootkit**)
  - Entender el file system y agregar/patchear un file (e.g **Computrace v.1**)

Ejecution / Time

- E.g Hookeando la “INT 13H” (REAL MODE)



No Common Ways

- Virtualizar el OS desde el principio
  - VMBR (Virtual Machine Based Rootkit)
    - E.g "**SubVirt**" (Univ. Michigan + Microsoft Research)

- Deep Boot ;-)

What is Deep Boot ?

- Es un proyecto (tecnica + research + implementacion)
- **Independiente del OS**
- Ejecucion Continua: NO pierde NUNCA el control de la ejecucion (durante el booteo)
- Usa **solo** features del Intel 80386 !
- Implementacion en C y ASM inlineado (16 y 32 bits)

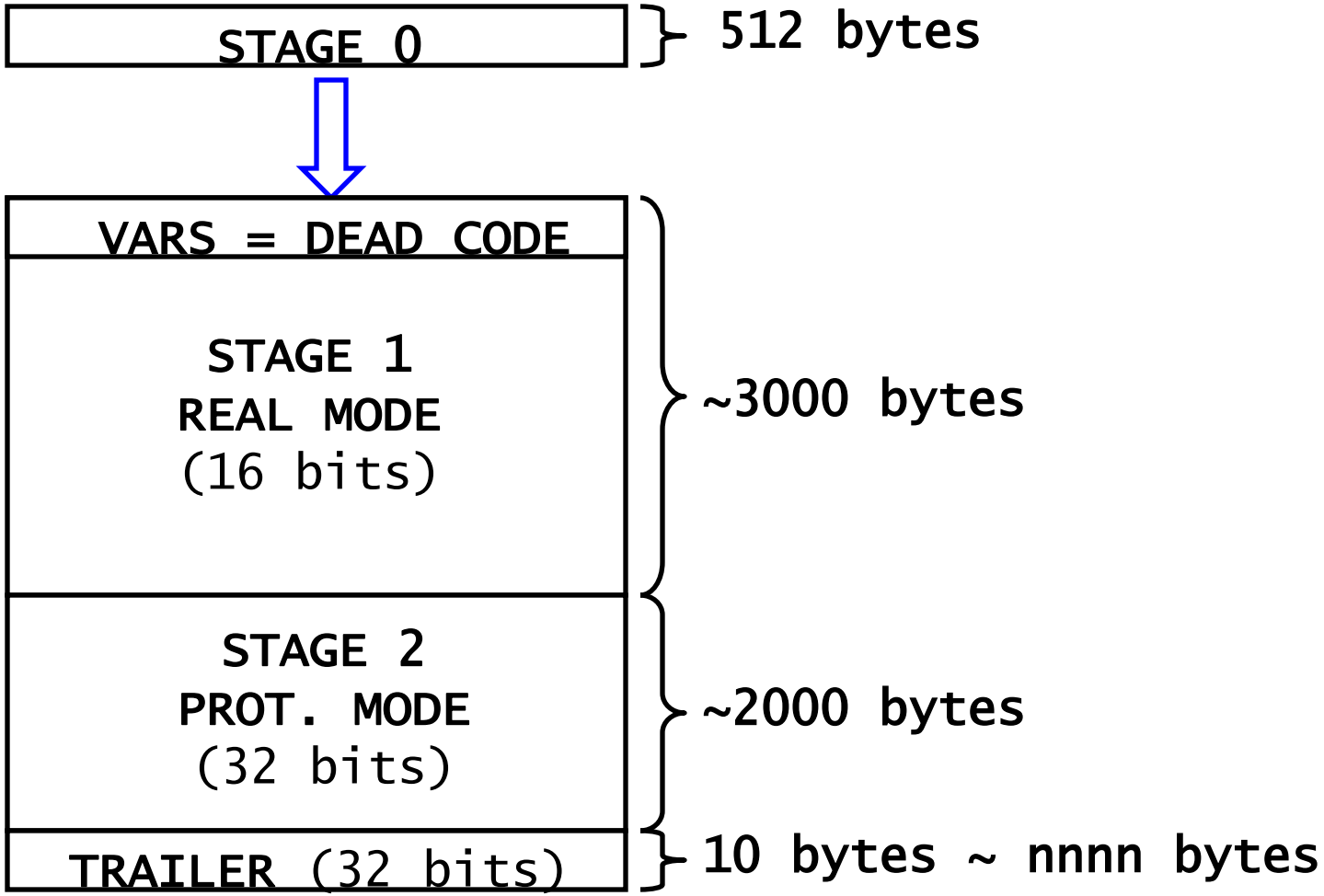
What is Deep Boot ?

- Se apoya sobre el mecanismo del TRAP FLAG (SINGLE STEP) para sobrevivir y controlar la ejecucion.
- “Emula” algunas instrucciones CRITICAS
- **Sobrevive a los cambios de contexto del OS !**
- NO AFECTA el funcionamiento del OS “victima”

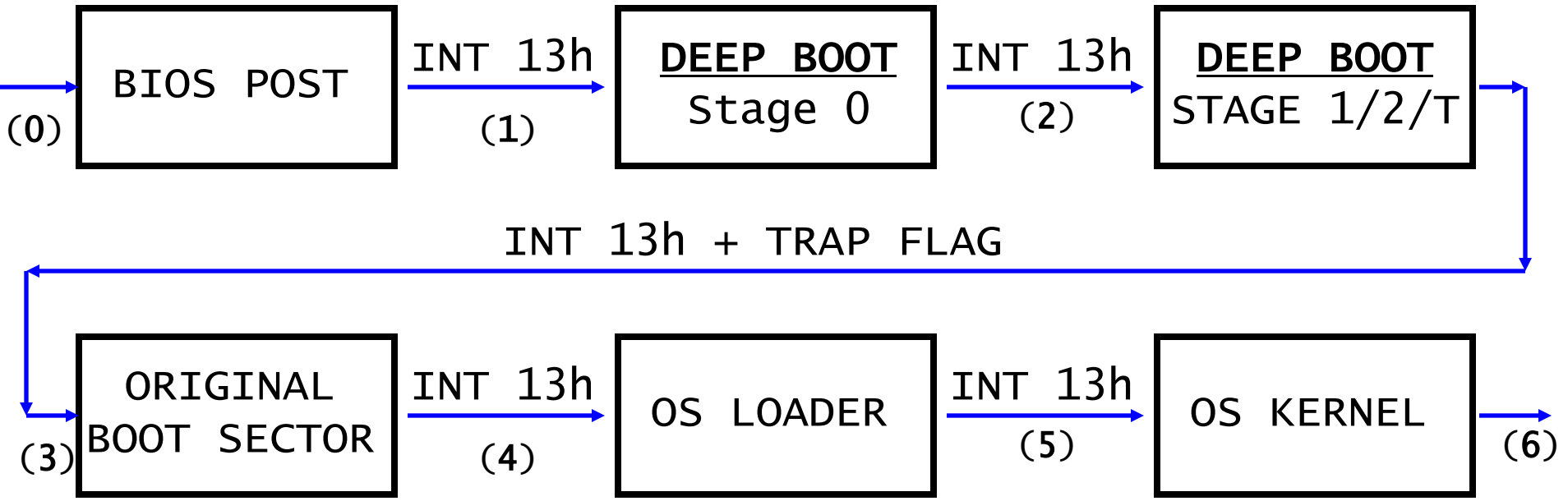
Deep Boot Body

- Compuesto por:
  - **Stage 0** (loader de stage 1 + stage 2 + trailer en MBR, pendrive, PXE, CD, FDD, etc)
  - **Stage 1** (Handler de 16 bits en REAL MODE)
  - **Stage 2** (Handler de 32 bits en PROT. MODE)
  - **Trailer** (CALLBACK de 32 bits llamado x cada instr. ejecutada en PROT. MODE)

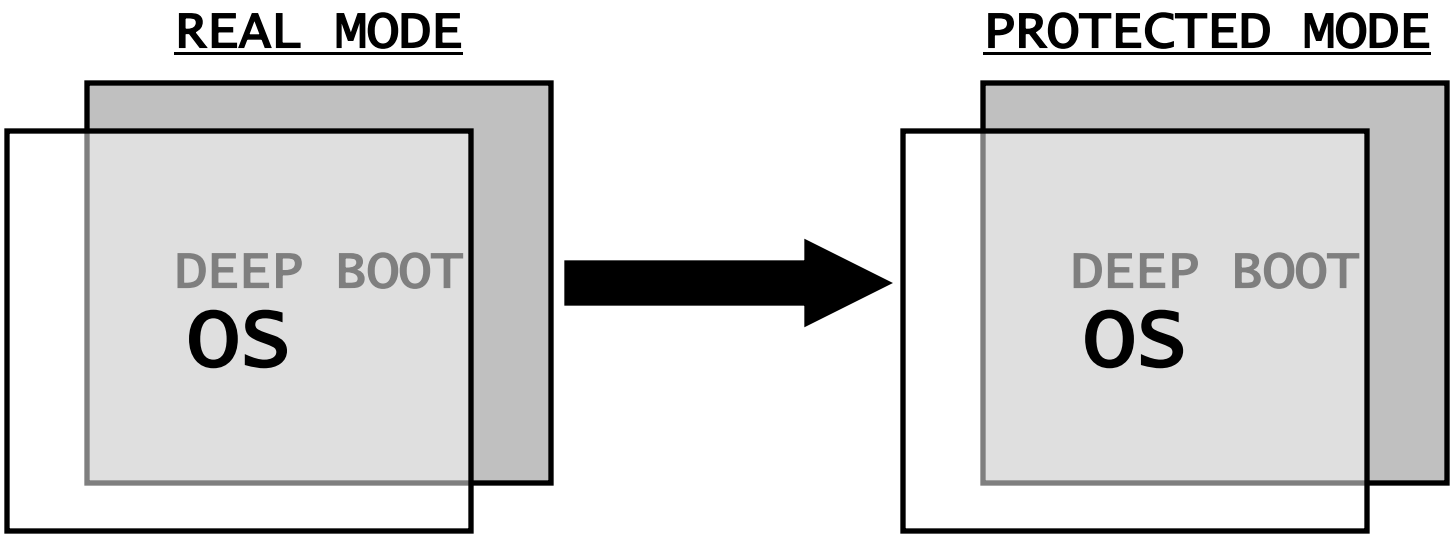
Deep Boot Body today



Taking Over from the beginning



Deep Boot Concept



A little of theory ...

- SINGLE STEP
- CPU Tables
  - GDT
  - IVT
  - IDT

SINGLE STEP

- Genera una excepcion por cada instrucción que se está por ejecutar
- La excepcion es procesada por un “handler” (callback)
- El handler se encuentra en la misma memoria fisica que la instrucción a ejecutarse (EL TRUCO !)
- Usado por todos los debuggers
- Implementado en x86-x64 en el registro EFLAGS (Trap Flag)

Intel TRAP FLAG

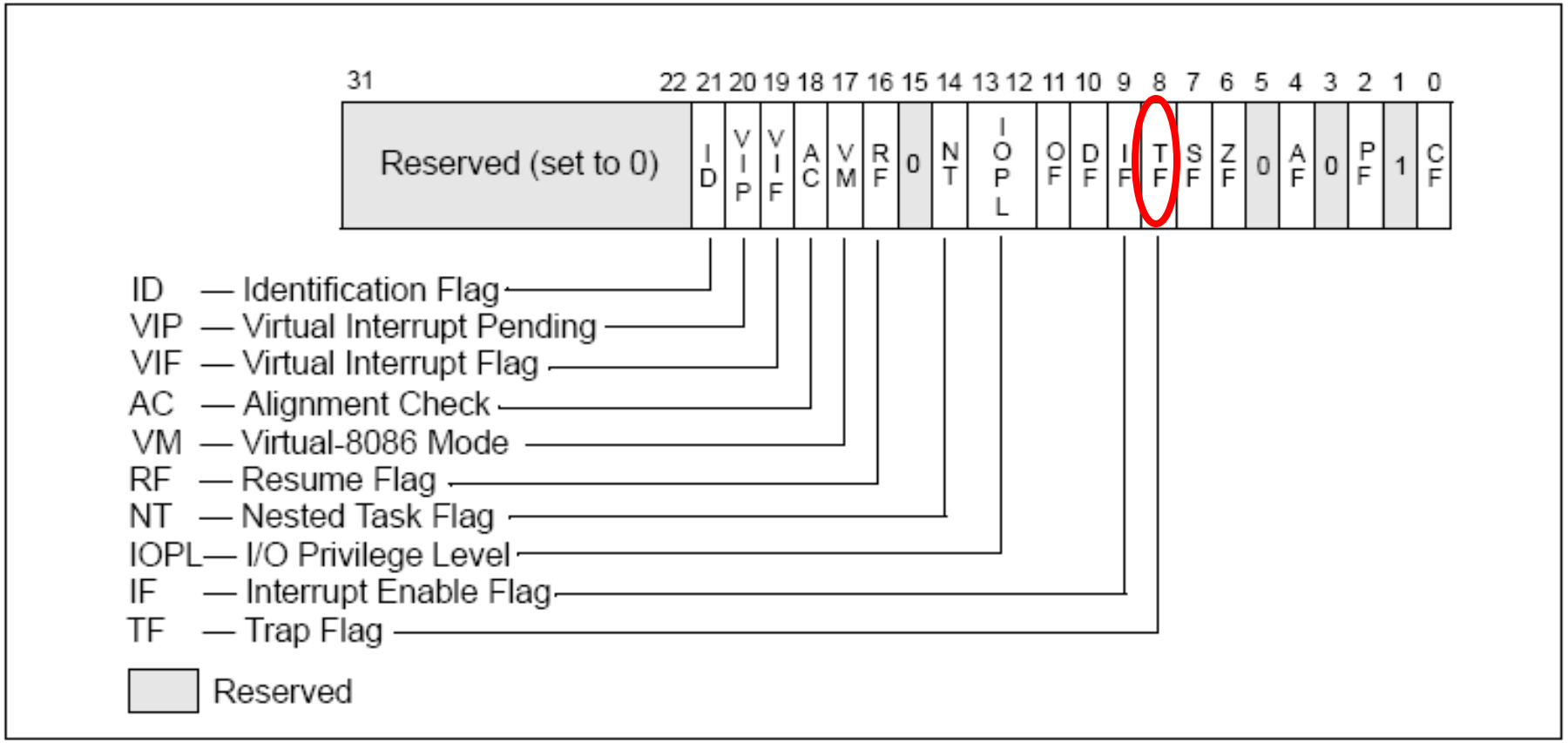
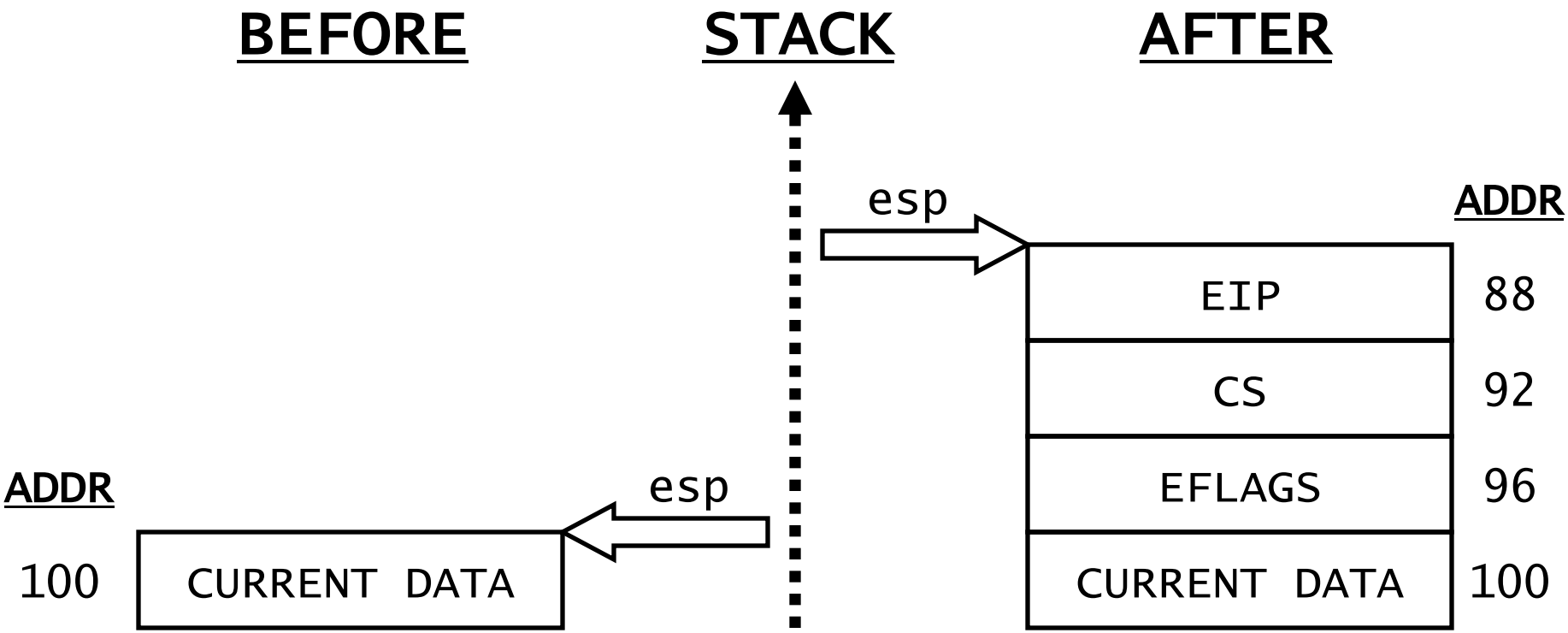


Figure 2-3. System Flags in the EFLAGS Register

Intel TRAP FLAG

- Cuando esta prendido:
  - Invoca a la interrupcion numero **1**
  - En REAL MODE direcciona mediante la IVT
  - En PROTECTED MODE direcciona mediante la IDT

Handling SINGLE STEPs



GDT

- GDT: “General Descriptor Table”
- Usada por el micro en **modo protegido**
- Sirve para “separar/proteger” areas de memoria
- Cada entrada de llama **DESCRIPTOR**
- El primer descriptor NO se usa (NULL)
- Longitud puede variar entre 3 a 8192 entradas.
- Se carga usando la instrucción “lgdt”

GDT DESCRIPTOR

- Define un segmento (area) de memoria
- Se los referencia con un offset en la GDT (**SELECTOR** – E.g CS, SS, DS, ES, FS,GS)
- Es una estructura de 8 bytes
  - » Base Address: DWORD (32 bits)
  - » Limit: WORD+NIBBLE (20 bits)
  - » Privileges: Code, Data, Gates

IVT

- IVT: “Interrupt Vector Table”
- Usada por el micro en **modo real**
- Sirve para asociar interrupcion-handler
- Interrupciones tambien son generadas por IRQs
- Cada entrada mide 4 bytes (SEGM:OFFSET)
- Longitud de 256 entradas.

IDT

- IDT: “Interrupt Descriptor Table”
- Usada por el micro en **modo protegido**
- Sirve para asociar interrupcion-handler
- Interrupciones tambien son generadas por IRQs
- Cada entrada de llama **DESCRIPTOR**
- Longitud puede variar entre 0 a 256 entradas.
- Se carga usando la instrucción “lidt”

IDT DESCRIPTOR

- Apunta a un handler (“callback”).
- Los usa el micro cuando se produce una interrupcion
  - E.g: → ZERO DIVISION → excepcion → el CPU lee el descriptor numero 0 de la IDT → int 0 → handler
- Es una estructura de 8 bytes
  - » Base Address: DWORD (32 bits)
  - » Selector: WORD (16 bits)
  - » Privileges: Ring desde donde puede ser llamado

Deep Boot Development

A hard work ...

Main Problems

- **Sobrevivir:**
  - **CONTEXT SWITCHS** del OS:
    - 1. REAL MODE → PROTECTED MODE
    - 2. PROTECTED MODE → PROTECTED MODE
    - 3. PROTECTED MODE → REAL MODE
  - **STACK SWITCHS** del OS
  - Mantenerse en un area de memoria confiable
  - Mantener la integridad del OS (MEM y REGS)
  - Mantener el TRAP FLAG PRENDIDO ...

REAL MODE → PROT. MODE

- Pasos normales:
  - 1. El OS carga una GDT (“lgdt”)
  - 2. El OS carga una IDT (“lidt”)
  - 3. El OS setea el PDE (registro CR3)
  - 4. El OS prende el bit 0-31 del registro CR0
  - 5. El OS finalmente ejecuta un JUMP FAR

REAL MODE → PROT. MODE

- Con Deep Boot corriendo:
  - El OS carga una GDT (“lgdt”)
  - El OS **INTENTA** cargar una IDT (“lidt”)
  - El OS setea el PDE (registro CR3)
  - El OS **INTENTA** prender el bit 0-31 del registro CR0
  - El OS generalmente ejecuta un JUMP FAR (pero esta vez en **REAL MODE** ;-))

REAL MODE → PROT. MODE

- Condiciones para el Deep Boot's CONTEXT SWITCH:
  - 1. Si el bit 0 del registro CR0 intentó ser prendido
  - 2. Si el OS hizo un JUMP FAR, RETF, CALL FAR, IRET
    - » If (LAST_CS != CURRENT_CS)

Crossing to PROT. MODE

- Pasos de Deep Boot:
  - 1. Agrega/reusa 2 descriptores de 32 bits en la GDT (codigo y datos)
  - 2. Setea 2 handlers en la **IDT original** (SINGLE STEP y BREAKPOINT)
  - 3. Carga la **IDT original** (la que no pudo el OS)
  - 4. Prende el bit 0-31 del registro CR0
  - 5. Hace un **IRETD** a **CURRENT_CS:CURRENT EIP** (Cruce efectivo a MODO PROTEGIDO)

PROT. MODE → PROT. MODE

- Pasos normales:
  - El OS carga una nueva GDT
  - El OS carga una nueva IDT
  - El OS habilita paginacion o cambia el PDE

- Con Deep Boot corriendo:
  - IDEM para el OS

PROT. MODE → PROT. MODE

- Pasos de Deep Boot:
  - Si el OS está por cargar una GDT (“lgdt”)
    - » **Deep Boot agrega/reusa 2 descriptores**
    - » **Deep Boot actualiza los IDT descriptors con la nueva GDT**
  - Si el OS está por cargar una IDT (“lidt”)
    - » **Deep Boot setea los 2 handlers en la nueva IDT (SINGLE STEP y BREAKPOINT)**
  - Si el OS cambia la base del PAGE DIRECTORY ENTRY (CR3)
    - » **Deep Boot no hace nada ... ???**

PROT. MODE → REAL MODE

- Pasos normales:
  - 1. El OS salta a codigo de 16 bits en PM
  - 2. El OS apaga el bit 0 del registro CR0
  - 3. El OS ejecuta un JUMP FAR
  - 4. El OS carga una nueva IDT apuntando a la direccion 0 (IVT)

PROT. MODE → REAL MODE

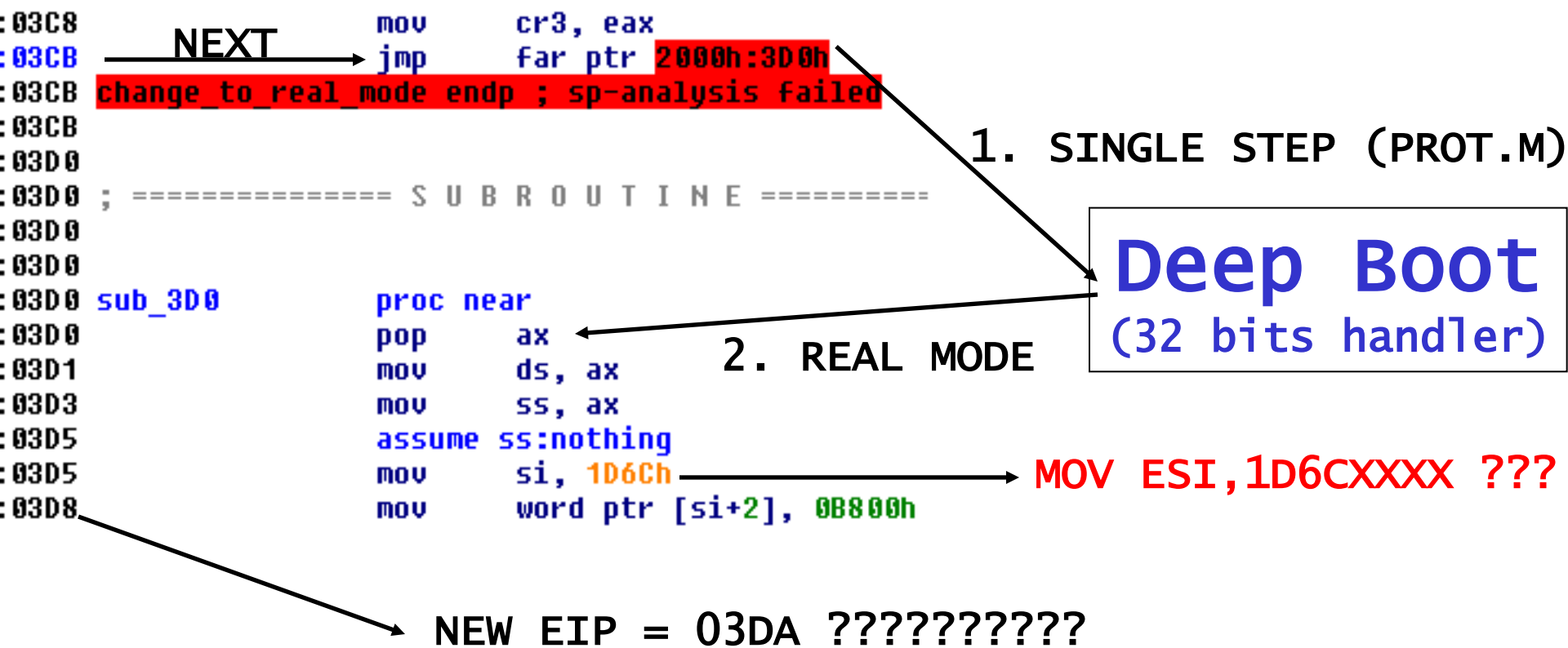
- Con Deep Boot corriendo:
  - 1. El OS salta a codigo de 16 bits (IDEM)
  - 2. Si el OS está por apagar el bit 0 del registro CR0
    - **Deep Boot setea una nueva IDT en 0 (IVT)**
  - 3. El OS apaga el bit 0 del registro CR0
  - 4. Se produce un “JUMP FAR” generado por el mismo SINGLE STEP (“int 1”)
  - 5. La interrupcion es catcheada por el handler en REAL MODE
  - 6. El handler cambia el CURRENT_CS por uno valido en REAL MODE

PROT. MODE → REAL MODE

- CS = selector de codigo de 16 bits
  - JUMP FAR to REAL_MODE → **OK**

- CS = selector de codigo de 32 bits
  - JUMP FAR to REAL_MODE → **HALT STATE ???**

Intel Undocumented Mode ?



Intel Undocumented Mode ?

- **REAL MODE** → 16 bits native code, 64 kb de direccionamiento de data.
- **UNREAL MODE** → 16 bits native code, hasta 4 GB de direccionamiento de data (using instruct. prefixes).
- **THIS MODE** (UNREAL MODE 32 ???)→ **32 bits native code** (without prefixes) + UNREAL MODE

OS Stack Switches

- **Recorte del Volumen 3 del manual de Intel**

5.6.3. Masking Exceptions and Interrupts When Switching Stacks

To switch to a different stack segment, software often uses a pair of instructions, for example:

```
MOV SS, AX
MOV ESP, StackTop
```

If an interrupt or exception occurs after the segment selector has been loaded into the SS register but before the ESP register has been loaded, these two parts of the logical address into the stack space are inconsistent for the duration of the interrupt or exception handler.

To prevent this situation, the processor inhibits interrupts, debug exceptions, and single-step trap exceptions after either a MOV to SS instruction or a POP to SS instruction, until the instruction boundary following the next instruction is reached. All other faults may still be generated. If the LSS instruction is used to modify the contents of the SS register (which is the recommended method of modifying this register), this problem does not occur.

OS Stack Switches

- Ejemplo:
  - SS1 BASE = 0, ESP1 = 0x4444
  - SS1:ESP1 → 0x0:0x4444 → 0x0 + 0x4444 → 0x4444
  - SS2 BASE = 0x70000, ESP2 = 0x1000
  - SS2:ESP2 → 0x70000:0x1000 → 0x70000 + 0x1000 → 0x71000
- Si el OS NO respeta la **ATOMICIDAD** ?
  - ? **SS2:ESP1** → 0x70000:0x4444 → 0x74444 → **BAD FOR US !** → MEMORY CORRUPTION → **BSoD**

OS Stack Switches

- NTLDR basic block (Windows)

```
00000997
00000997 loc_997:                        ; - VIDEO - SET CURSOR POSITION
00000997 int     10h                     ; DH,DL = row, column (0,0 = upper left)
00000997                                 ; BH = page number
00000999 pop     bp
0000099A add     sp, 8
0000099D push    1
0000099F call    near ptr change_to_protected_mode
000009A2 add     sp, 2
000009A5 pop     ebx
000009A7 mov     dx, 10h
000009AA mov     ds, dx
000009AC assume ds:nothing
000009AC mov     ss, dx
000009AE assume ss:nothing
000009AE mov     es, dx
000009B0 assume es:nothing
000009B0 mov     esp, ebx
000009B3 pop     edi
```

INCONSISTENT STACK AREA

OS Stack Switches

- Deep Boot Solution:
  - 1.Si el OS va a ejecutar un “mov ss,algo”
    - »Deep Boot apunta EIP a la proxima inst. y se pone en alerta
  - 2. Si el OS va a ejecutar un “mov esp,valor”
    - »Deep Boot salta a un trampoline para switchear al nuevo **STACK**

BIOSPHERE

- Area de memoria donde se ejecuta Deep Boot.
- Direccion Base = 9E00h:0000h = 0x9E000
- Dentro del primer MEGABYTE
  - » En REAL MODE
  - » En PROTECTED MODE
- Codigo y variables contenidos en la misma area.
- Memoria shareada con el OS (comparten el mismo contexto).

OS Integrity

- Guardar y Restaurar el estado de todos los registros en cada SINGLE STEP handleado.
- No escribir memoria que esté fuera de la BIOSFERA
- No alterar el **curso normal** de la ejecucion

If the OS doesn't need IDT ?

- En las primeras etapas, el OS puede NO necesitar una IDT (Ej. GRUB)
- Deep Boot necesita **si o si** una IDT para funcionar
- Si el OS intenta pasar a modo protegido sin IDT
  - Deep Boot crea una IDT **temporal** propia

Misc

- Evitar que el TRAP FLAG sea apagado ?
  - Ningun OS intentó apagarlo ...
- Para poder correr codigo en C
  - Setear DS = SS y EBP = ESP
- Codigo PIC en C
  - No usar ptr a funcions, strings ni variables globales
  - Hubo que crear la funcion “get_pic_address()”

"Emulated/Intercepted" Instructions

- LGDT
- LIDT
- MOV CR0, REG32
- MOV SS, REG16
- MOV ESP, REG32
- JUMP LARGE FAR (RM → PM 32)

Performance

- Cada **instrucción traceada** tiene un costo
- Promedio de 700.000 inst/seg (en esta notebook)
- La idea es eliminar la mayor cantidad de handleos **sin perder el control**

Optimization Chances

- Emular algunas instrucciones basicas
  - » jump condicionales
  - » jump incondicionales
  - » mov reg32,reg32
  - » nops
- Usando Breakpoints
  - Chequear el destino y saltear algunas instrucciones de repeticion (“rep movsX”, “rep stosX”, “rep outsX”)
  - Bypassear llamados a funciones (PELIGROSO !)

Deep Boot Applications

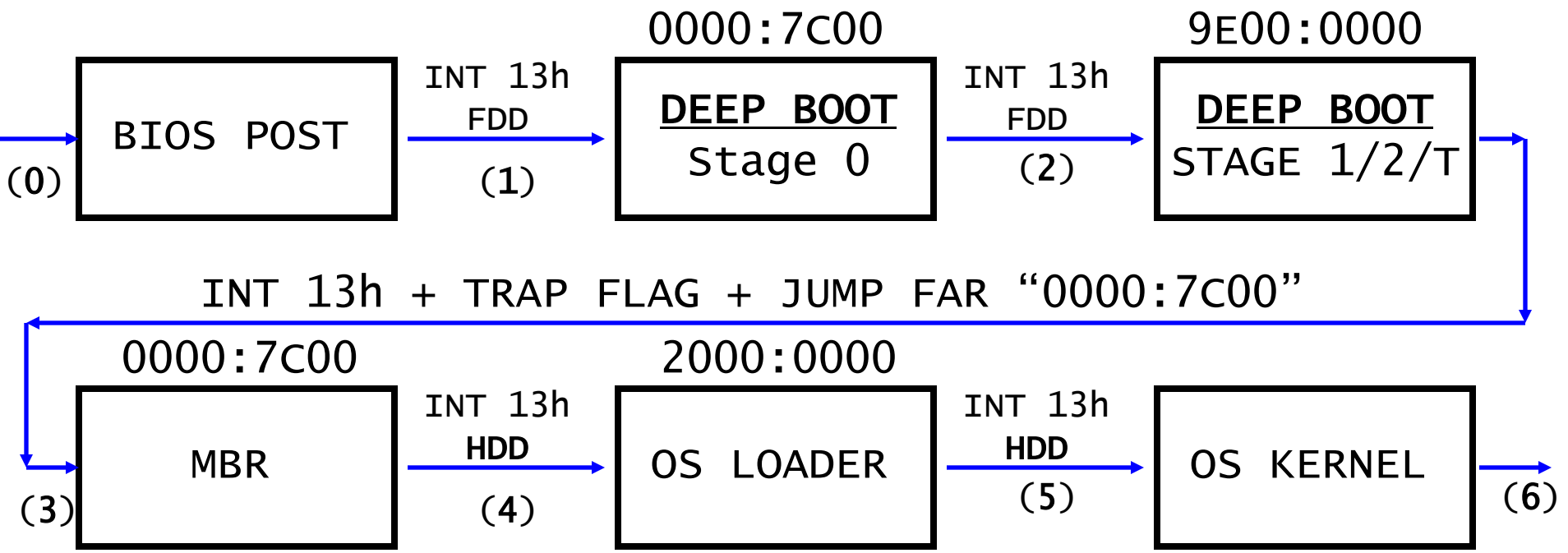
- Rootkits (?)
- Hot Patching (bugs en kernel)
- Bypasses en el codigo (e.g overwrite the kernel “setuid” function)
- Agregarle nuevas funcionalidades al OS
- Interaccion con el OS (Uso de funciones)
- Encapsulamiento del OS (Hypervisor)
- Debugger de kernel generico desde el booteo

DEMO TIME

Deep Boot

Trailer

Booting from a Removable Device



DEMO 1

- Victima:
  - OpenBSD v4.5

- Boot Device:
  - Virtual FDD
- Condicion de Corte:
  - 30.000.000 instrucciones ejecutadas
- Proposito:
  - Demostrar el funcionamiento de Deep Boot

DEMO 2

- Victima:
  - Arch Linux - kernel v2.6.35

- Boot Device:
  - Virtual FDD
- Condicion de Corte:
  - Cuando la funcion **getuid()** es patcheada
- Proposito:
  - Setear el **ID 0** a todos los usuarios del sistema

DEMO 2 – patching getuid()

```
mov eax, fs:[0xc144048c]
mov eax, [eax+1FCh]
    push ebx
    xor ebx,ebx
    mov [eax+4],ebx
    pop ebx
mov eax, [eax+4]
ret
```

PATCH

DEMO 3

- Victima:
  - Windows XP SP3 con Kaspersky 2011

+

- Boot Device:
  - HDD (Master Boot Record)
- Condicion de Corte:
  - Cuando se ejecuta la funcion “ntkrlnpa.KiSystemStartup()”
- Proposito:
  - Colgar en memoria el **ROOTKIT** instalado previamente

Supported OSes (until now)

- Windows
  - XP
  - 2003
  - Windows Vista
  - Windows 7
  - Windows 2008 (32 y 64 bits)
- Linux
  - Debian 6.0
  - Arch
- OpenBSD

What is the MBR ?

- MBR (Master Boot Record)
  - Primer sector del disco rigido
  - Mide 512 bytes
  - Contiene el stage 0 (primer loader) de cualquier OS
  - Contiene la tabla de particiones

```
0118  13 61 61 73 0E 4F 74 0B  32 E4 8A 56 00 CD 13 EB  .aas.Ot.2S.V.-.d
0128  D6 61 F9 C3 49 6E 76 61  6C 69 64 20 70 61 72 74  +a-+Invalid.part
0138  69 74 69 6F 6E 20 74 61  62 6C 65 00 45 72 72 6F  ition.table.Erro
0148  72 20 6C 6F 61 64 69 6E  67 20 6F 70 65 72 61 74  r.loading.operat
0158  69 6E 67 20 73 79 73 74  65 6D 00 4D 69 73 73 69  ing.system.Missi
0168  6E 67 20 6F 70 65 72 61  74 69 6E 67 20 73 79 73  ng.operating.sys
0178  74 65 6D 00 00 00 00 00  00 00 00 00 00 00 00 00  tem............
0188  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0198  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01A8  00 00 00 00 00 00 00 00  00 00 00 00 00 2C 44 63  .............,Dc
01B8  C2 B4 C2 B4 00 00 80 01  01 00 07 FE BF 08 3F 00  -¦-¦.......¦+.?.
01C8  00 00 8A B6 7F 00 00 00  00 00 00 00 00 00 00 00  ...¦............
01D8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01E8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
01F8  00 00 00 00 00 00 55 AA                           ......U¬........
```

Protections ?

- BIOS + TPM (Chip)
  - BitLocker (Microsoft)
    - » Windows Vista Ultimate/Enterprise
    - » Windows 7 Ultimate/Enterprise
    - » Windows 2008
  - TrustedGRUB
    - » Linux

HLT

- Questions ?