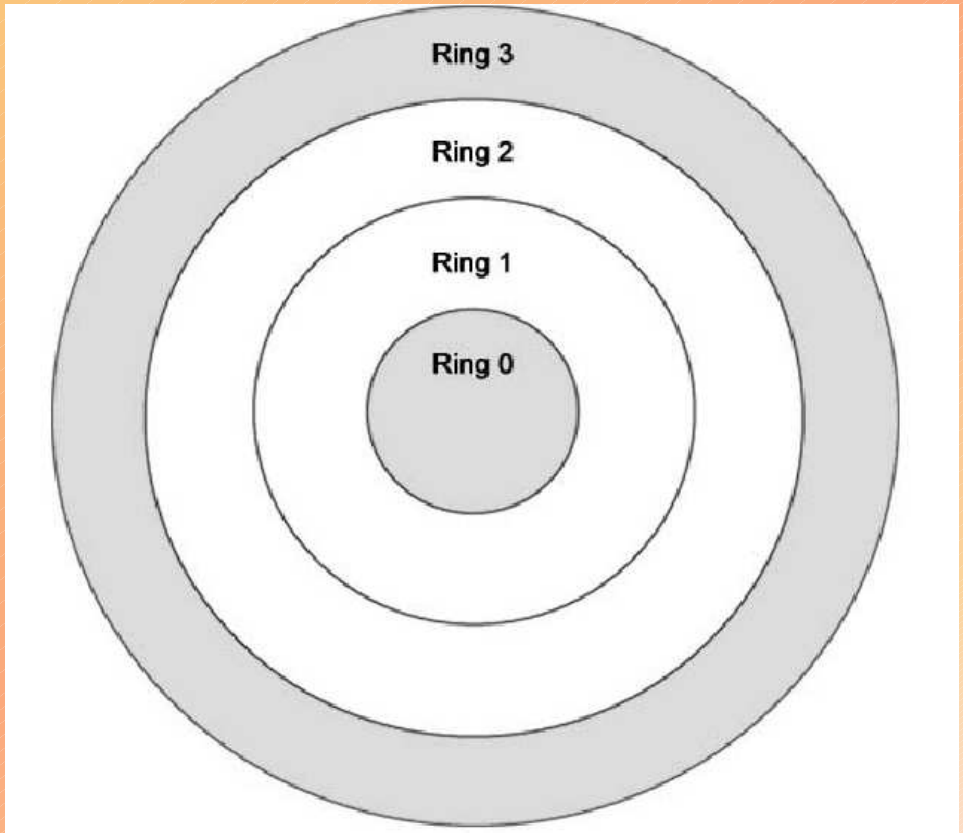
DKOM
(Direct Kernel Object Manipulation)

Sistema Operativo: Struttura (1)

- User Mode
  - Il sistema operativo fornisce API ai programmatori
    - Kernel32.dll
    - Ntdll.dll
- Kernel Mode
  - Funzioni a basso livello che implementano servizi richiesti dal user mode
  - Aree memoria protetta contenenti oggetti come processi, tokens, porte, ecc…

Sistema Operativo: Struttura (2)

- Intel ha progettato quattro livelli di privilegio chiamati *rings*
- Microsoft e molti altri sviluppatori di Sistemi Operativi hanno preferito implementare solamente due *rings*



Sistema Operativo: Struttura (3)

- Usando solo due livelli di privilegio, non c’è separazione tra il kernel stesso ed i drivers di terze parti o i moduli del kernel (loadable kernel midules o LKM)
- I drivers possono modificare la memoria associata agli oggetti kernel come per esempio un *token* di processo

HIDS / HIPS (1)

- HIDS = Host Intrusion Detection Systems
- HIPS = Host Intrusion Prevention Systems

- Utili per rilevare o prevenire
  - Processi in esecuzione
  - Creazione, modifica ed eliminazione di files
  - Connessioni di rete
  - Scalata di privilegi
- I sistemi HIPS / HIDS si affidano al sistema operativo per le loro attività
- Se il sistema operativo sottostante è compromesso, di conseguenza i sistemi HIDS / HIPS falliscono !!!

HIDS / HIPS (2)

- Come funzionano gli HIDS / HIPS ?
  - Interrogazione delle funzioni del kernel
  - *Hooking* delle suddette funzioni API (kernel32.dll, ntdll.dll)
  - Hooking della tabella della System Calls
  - Registrazione delle funzioni di call-back fornite dal Sistema Operativo

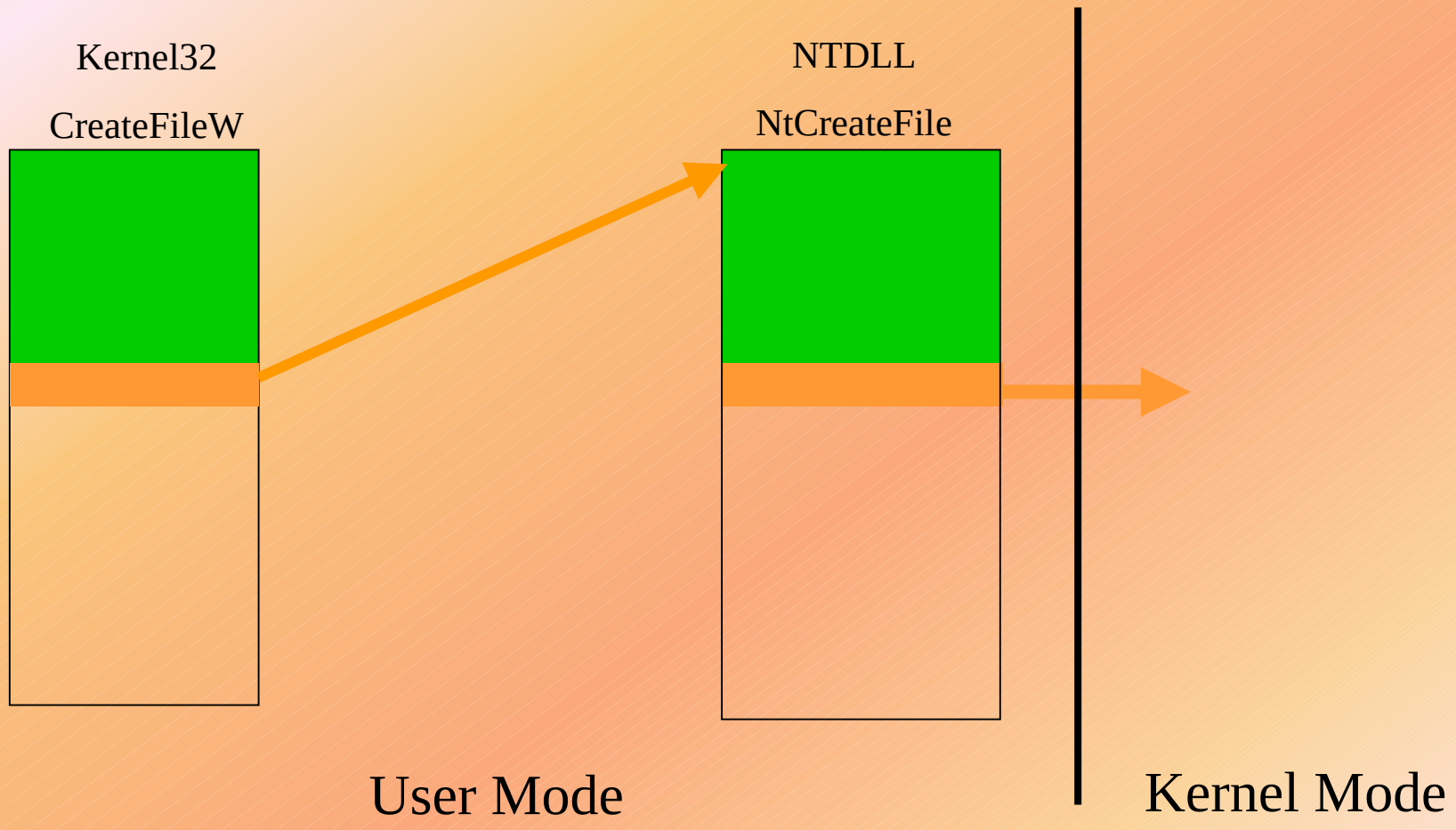
Scenario d’Attacco

- L’attaccante ottiene diritti di privilegio elevati
- L’attaccante installa un *rootkit*
- Funzioni del *rootkit:*
  - Nascondere processi
  - Nascondere files
  - Nascondere connessioni di rete
  - Installare una *backdoor* per accedere in seguito al sistema
- I rootkits agiscono come parte del Sistema Operativo, in modo da avere accesso allo spazio di memoria del kernel

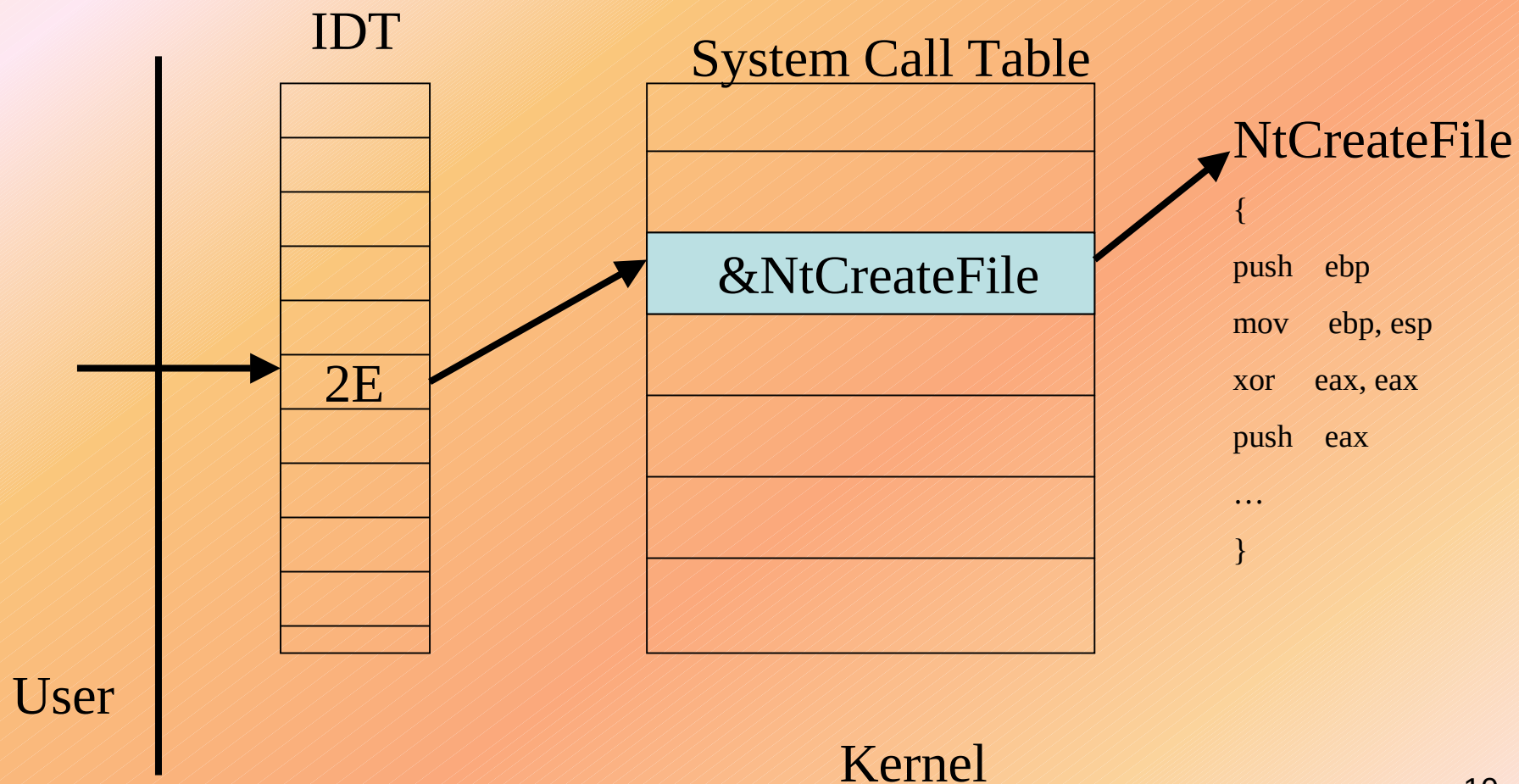
Rootkits: Stato Corrente

- In un primo momento, i *rootkits* non erano molto più dei normali *Trojans*
- Ma…i *rootkits* evoluti **filtrano** i dati
  - Agganciano (hook) la System Call Table del Sistema Operativo (ovvero la tabella che contiene le funzioni esportate dal kernel)
  - Agganciano (hook) l' Interrupt Descriptor Table (IDT)
    - Gli Interrupts sono per segnalare al kernel che c'è un certo compito da eseguire
    - Aggangiando un interrupt, un rootkit evoluto può filtrare tutte le funzioni esportate dal kernel

Flusso di Controllo (1)



Flusso di Controllo (2)



DKOM (1)

- **Direct Kernel Object Manipulation (DKOM) in memory**
  - Un device driver o un LKM ha accesso alla memoria del kernel
  - Un rootkit sofisticato può modificare gli oggetti direttamente in memoria in una modalità nascosta
  - Infatti.. Il principale obiettivo dei rootkit è quello di nascondere le cose: processi, files e connessioni di rete

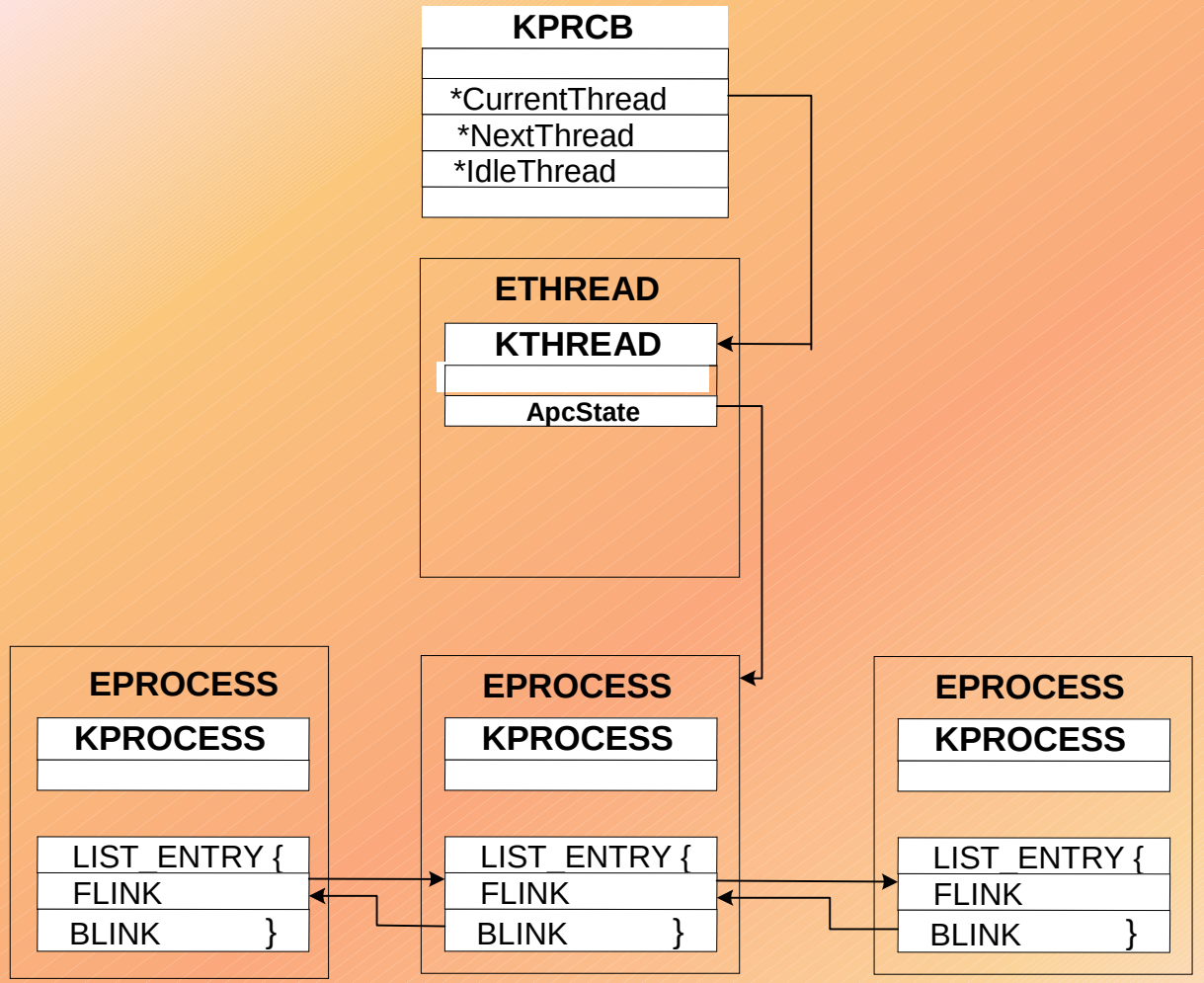
DKOM (2)

- DKOM:
  - Nasconde processi
  - Aggiunge privilegi ai tokens
  - Aggiunge gruppi ai tokens
  - Manipola i tokens per falsare la registrazione dell’Event Viewer di Windows
  - Nasconde porte

Processi Nascosti: Implicazioni

- L’intruso ottiene il pieno controllo del sistema
- Sconfitta dei sistemi HIDS / HIPS che si basano su quanto a loro fornito dal Sistema Operativo
- Vanificazione dei risultati della *forensic analysis*

Processi Nascosti: Windows (1)



Processi Nascosti: Windows (2)

- Localizzare il *Processor Control Block (KPRCP)*
  - posizionato all'indirizzo 0xffdff120
  - Il registro fs punta all'indirizzo 0xffdff000
- All'interno del KPRCB c'è un puntatore al *Current Thread Block (ETHREAD)*
  - Posizionato a fs:[124] o 0xffdff124
  - Un ETHREAD contiene una struttura *KTHREAD*

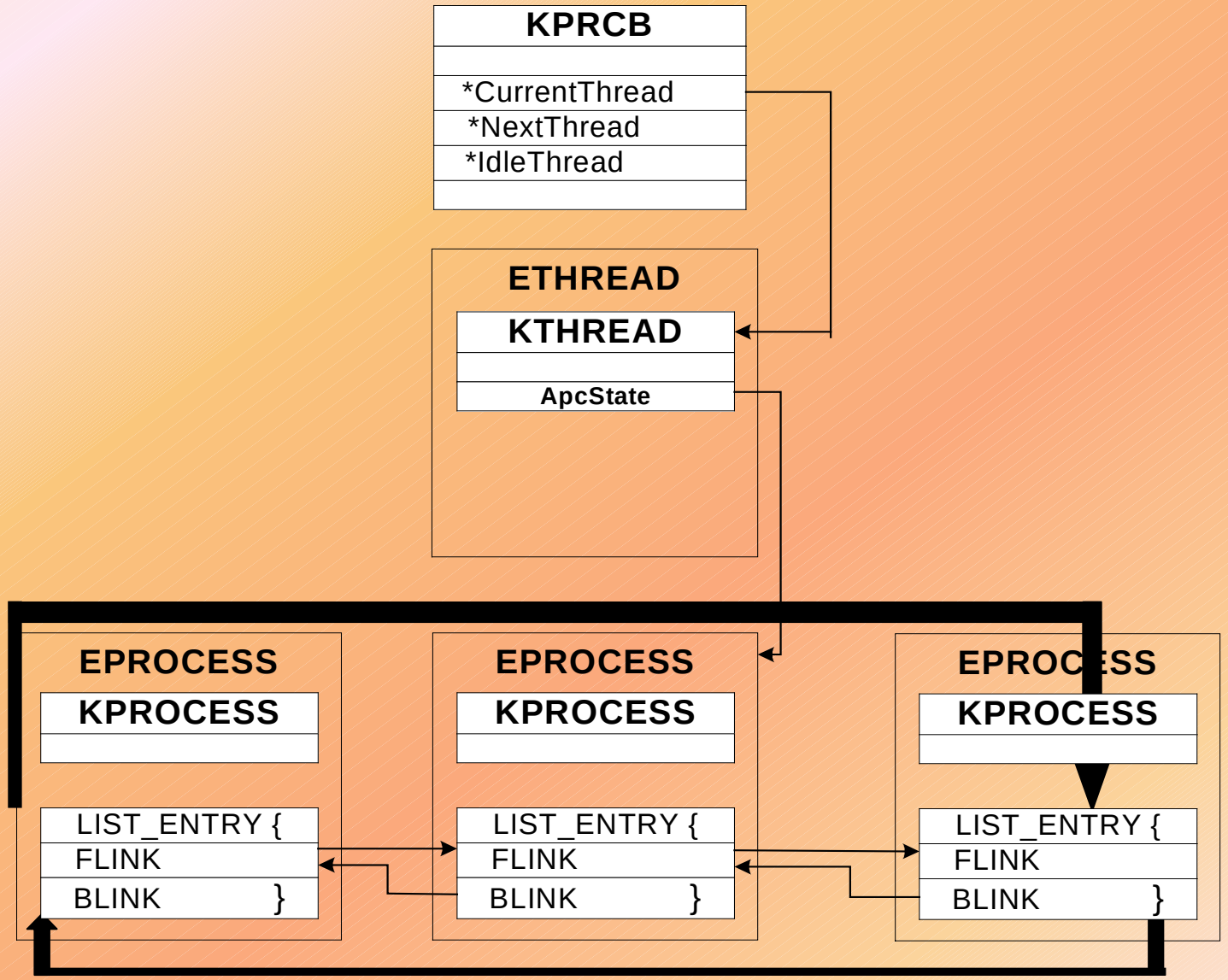
Processi Nascosti: Windows (3)

- La struttura KTHREAD contiene un puntatore al blocco *EPROCESS* del processo corrente
- Il blocco EPROCESS contiene una struttura LIST, che ha un puntatore in uscita ed uno in entrata al processo attivo
  - Questo crea la doppia *linked-list* dei processi attivi in Windows

Processi Nascosti: Windows (4)

- Per nascondere un processo:
  - Localizzare il blocco EPROCESS del processo da nascondere
  - Cambiare il processo precedente a quello da nascondere in modo che punti al processo successivo a quello da nascondere
  - Cambiare il processo successivo a quello da nascondere in modo che punti a quello precedente a quello da nascondere
- Ovvero, la doppia linked-list dei processi attivi salta il processo da nascondere e gli gira intorno

Processi Nascosti: Windows (5)



Processi Nascosti: Windows (6)

- Perché il processo nascosto rimane in esecuzione ?
  - Lo scheduling dell'esecuzione in Windows è basata sui threads e non sui processi
- Sebbene l'operazione di scheduling sia basata sui threads, quando il kernel riporta cosa sia in esecuzione in un certo momento, si basa sui blocchi EPROCESS, i quali possono essere modificati senza particolari effetti negativi.

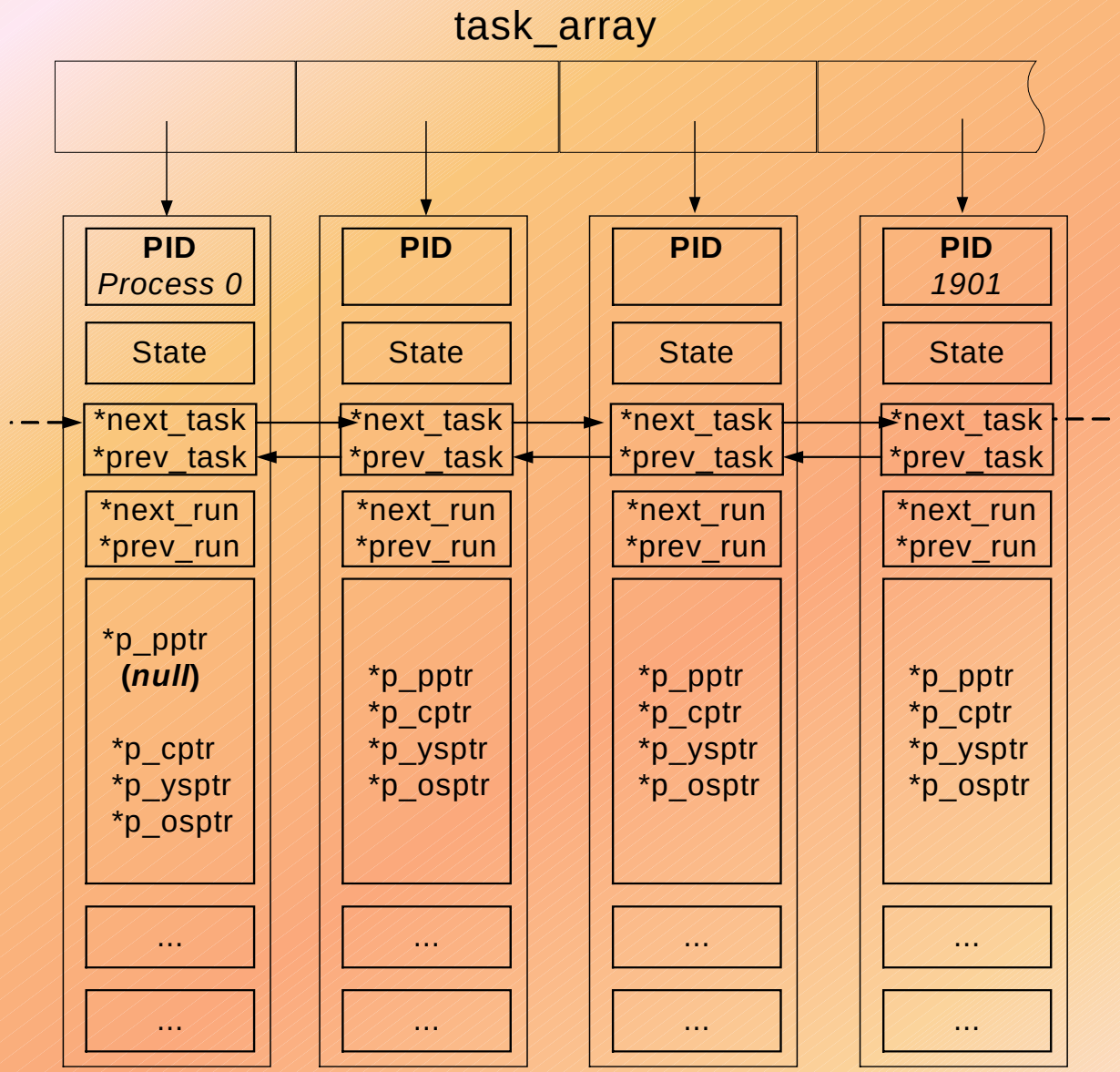
Processi Nascosti: Linux (1)

- Il kernel di Linux contiene un array di *task_struct*
- Una *task_struct* è simile ad un blocco EPROCESS dei sistemi Windows
- Una *task_struct* contiene puntatori a un *prev_task* e ad un *next_task*
- Una *task_struct* contiene anche puntatori al *prev_run* e al *next_run* per i processi in esecuzione

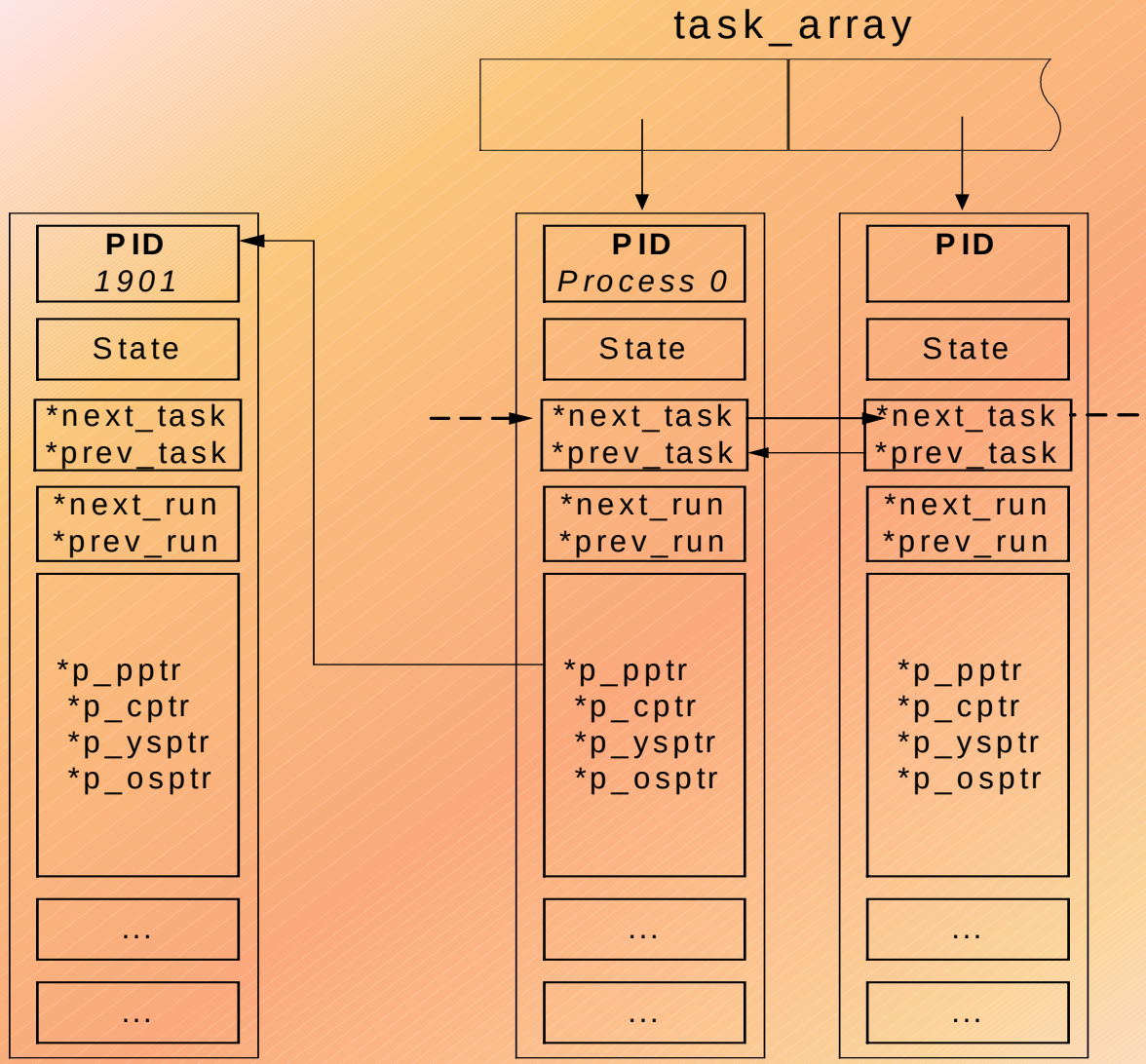
Processi Nascosti: Linux (2)

- Per nascondere un processo:
  - Rimuovere il processo dalla lista del *prev_task* e del *next_task*
  - Lasciare isolati il *next_run* ed il *prev_run*

Processi Nascosti: Linux (3)



Processi Nascosti: Linux (4)



Processi Nascosti: Linux (5)

- Per evitare che il processo nascosto si blocchi
  - Lo scheduler di Linux esamina la lista del *task_struct* per decidere se eseguire o meno il processo
  - Lo scheduler di Linux deve essere modificato per allocare tempi di esecuzione (quantums) al *padre* di un processo con PID 0

Tokens: Manipolazione (1)

- Aggiungere privilegi ai tokens
- Aggiungere gruppi ai tokens
- Cambiare l'*owner* del token
- Falsificare l'identità di chi ha eseguito un certo processo, sostituendola con una fittizia (ad esempio System)
  - Rende difficile la forensic analysis
  - Falsa completamente l'attendibilità dell'Event Viewer

Tokens

- **Parte Statica**
  - TOKEN SOURCE
  - TokenId
  - AuthenticationId
  - PartentTokenId
  - ExpirationTime
  - TokenLock
  - ModifiedId
  - SessionId
  - UserAndGroupCount
  - RestrictedSidCount
  - PrivilegeCount
  - VariableLength
  - Etc
- **Parte Variabile**
  - Privilegi
    - LUID
    - Attributi
  - Utenti e Gruppi
    - Puntatore al SID
    - Attributi
  - Restricted SID
    - Puntatore al SID
    - Attributi

Tokens: Manipolazione (2)

- E’ difficile modificare il token espandendolo, perché non si conosce che cosa ci sia in memoria dopo la parte variabile
- Sebbene la parte statica abbia puntatori a privilegi e gruppi, il solo cambiamento di questi in modo che puntino ad altre aree di memoria, non è sufficiente a causa della complessità dei calcoli operati dalla funzione SepDuplicateToken()

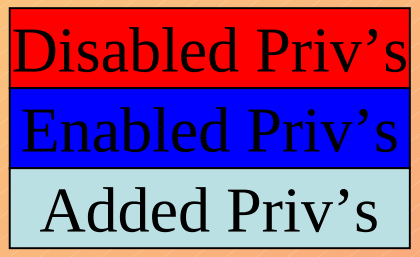
Tokens: Manipolazione (3)

- Ci sono molti privilegi disabilitati in un token
- E’ possibile disfarsi di questi privilegi in quanto disabilitati e liberare così spazio in memoria per nuovi privilegi e gruppi
  - Metodo “in-line”

Aggiungere privilegi ai token: DKOM (1)

- Typedef struct _LUID_AND_ATTRIBUTES{
  DWORD Luid;
  DWORD Attributes;
  }

Aggiungere privilegi ai token: DKOM (2)

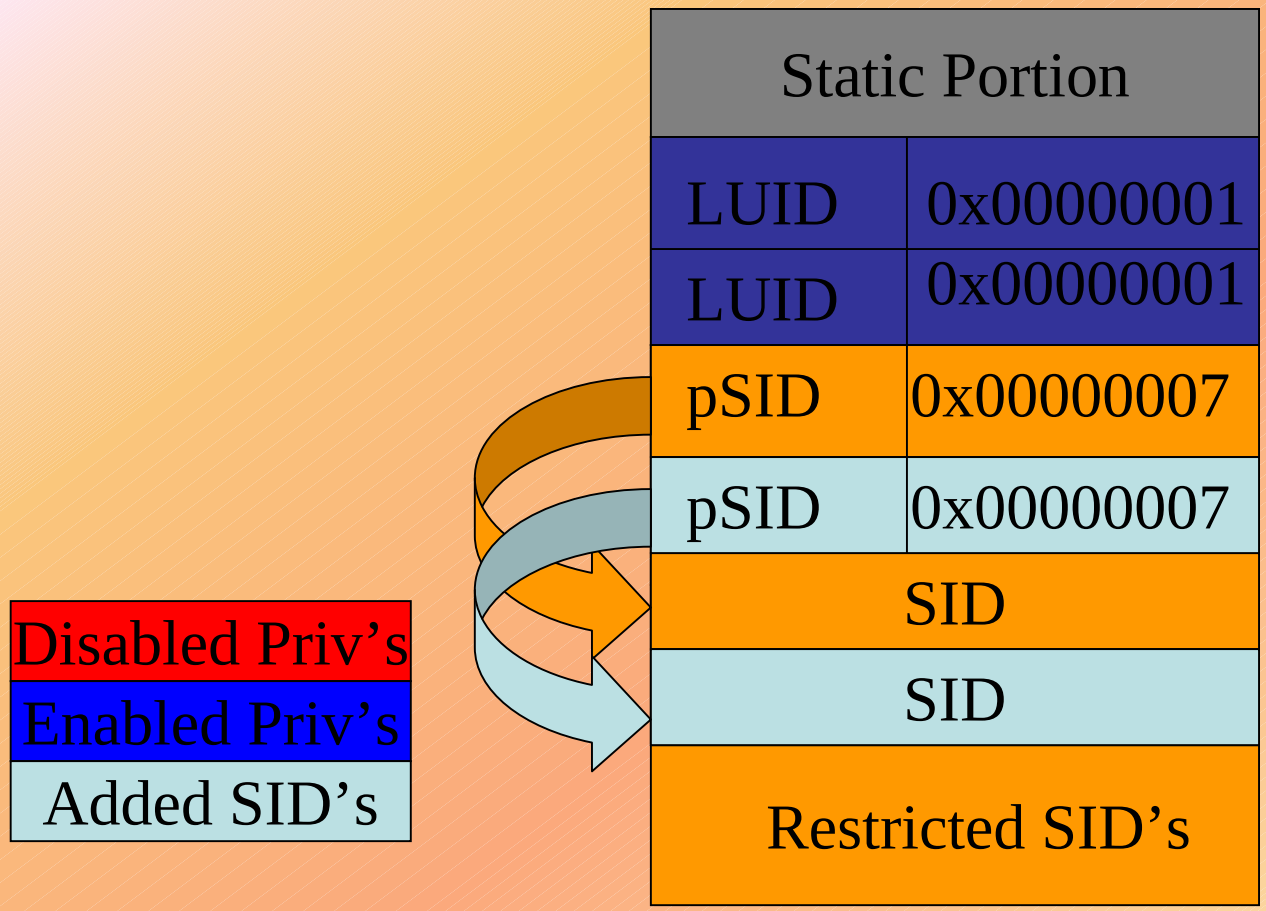


Static Portion	
LUID	0x00000001
LUID	0x00000001
LUID	0x00000001
LUID	0x00000001
SID's	
Restricted SID's	

Aggiungere privilegi ai token: DKOM (3)

- Typedef struct _SID_AND_ATTRIBUTES
  {
  DWORD pSID;
  DWORD Attributes;
  }

Aggiungere privilegi ai token: DKOM (4)



DKOM: Event Viewer

- Cambiare una DWORD nella parte statica del token
  - SYSTEM_LUID = 0x000003E7
- Poni il FIRST SID nel token al posto del System SID
- Tutte le registrazioni dei processi verranno fatte a carico di System
- Utile se è abilitato il Detailed Process Tracking

Processi Nascosti: Rilevazione

- Esaminare ogni thread per assicurarsi che il suo *process descriptor* corrispondente (EPROCESS) sia opportunamente *linkato*
- Queste operazioni richiedono il patching del kernel in memoria, in particolare per quanto riguarda la funzione SwapContext()
- Hunt e Brubacher hanno sviluppato Detours per intercettare le funzioni Win32

Il Patching del Kernel di Windows

- La funzione SwapContext() non cambia il contesto tra i threads in Windows
- Sovrascrivere i primi sette bytes della funzione SwapContext() con un jump alla funzione Detour
- Il registro EDI punta al KTHREAD del thread pronto per essere eseguito
- La funzione Detour segue il KTHREAD fino al blocco EPROCESS e determina questo è ancora correttamente linkato nella lista dei processi attivi