ROOTKIT

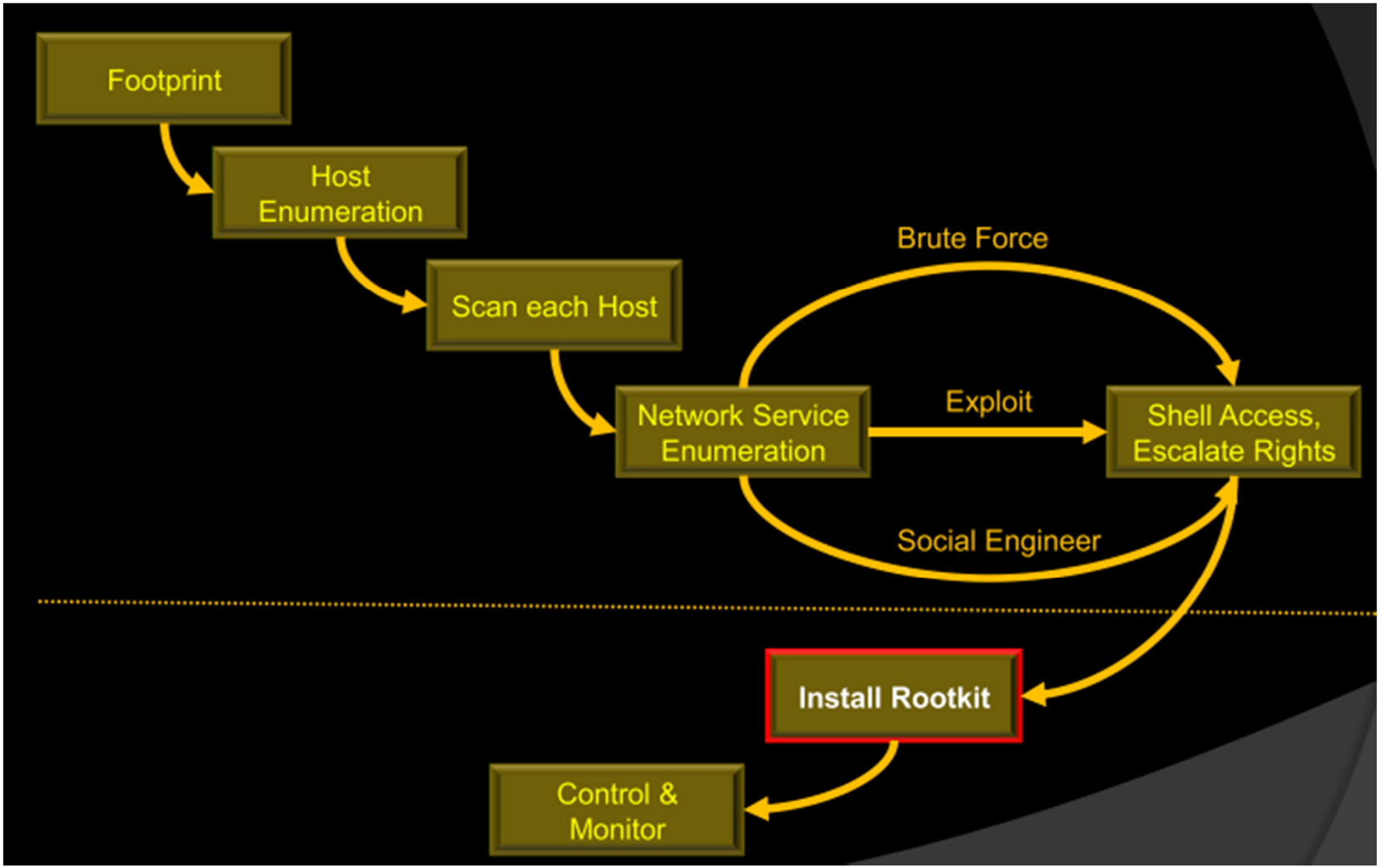
Rootkit là gì?

- Rootkit là một "kit" gồm các chương trình nhỏ và hữu dụng cho phép attacker duy trì việc truy xuất đến root (user có quyền lực nhất trên máy tính).
- Rootkit là một tập chương trình cho phép duy trì sự hiện diện lâu dài và chắc chắn trên máy tính mà không bị phát hiện bởi các phương tiện quản trị và an ninh thông thường.

Các dịch vụ được cung cấp bởi Rootkit

- Command and Control (C2)
- Surveillance (giám sát thu thập)
- Concealment (che giấu)

Rootkit là Post-Intrusion Tool



Rootkit hoạt động như thế nào?

- Rootkits làm việc dựa trên phương thức sửa đổi.
- Xác định và sửa đổi các phần mềm khiến cho chúng đưa ra các quyết định sai lầm.
- Có nhiều chỗ có thể sửa đổi trong phần mềm.

Patching

- Software logic có thể bị thay đổi nếu các data byte bị sửa đổi. Kỹ thuật này được gọi là patching
- Byte patching là một trong những kỹ thuật chính được dùng bởi các hoạt động bẻ khóa phần mềm.

Easter Eggs

- Sự thay đổi software logic có thể được thiết lập một cách cố ý từ đầu.
- Người lập trình có thể đặt một back door trong chương trình do họ viết ra.
- Back door này không được trình bày trong tài liệu và nó là đặc tính bị giấu.
- Dạng này được gọi là Easter Egg.
- Có thể được dùng như chữ ký

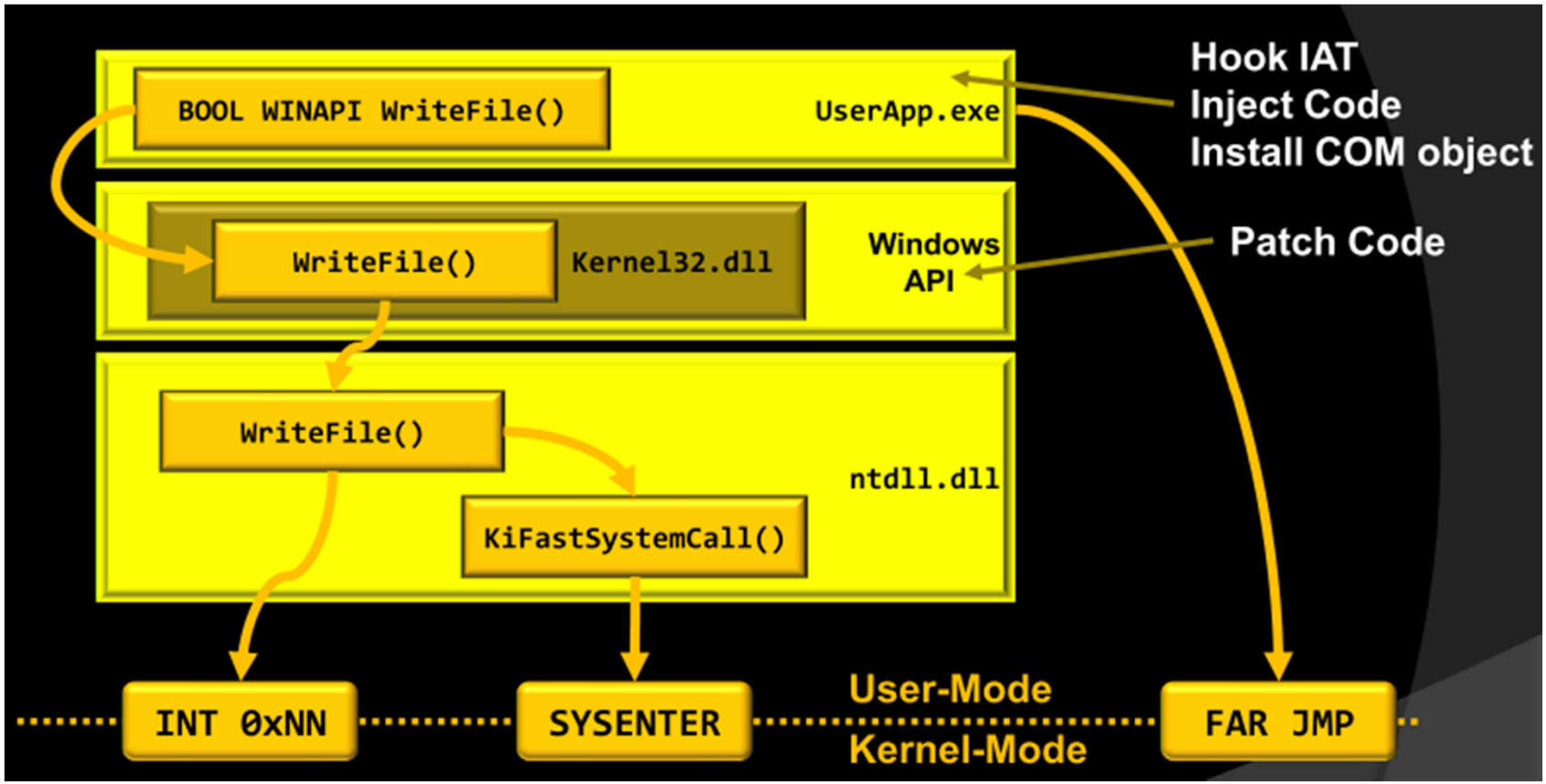
Spyware Modifications

- Một chương trình sẽ sửa đổi chương trình khác để chèn vào một spyware. Ví dụ spyware theo dõi các website là người dùng truy cập.
- Khó phát hiện spyware
- Ví dụ: spyware đánh lừa các trình duyệt hay shell, rất khó xóa chúng.

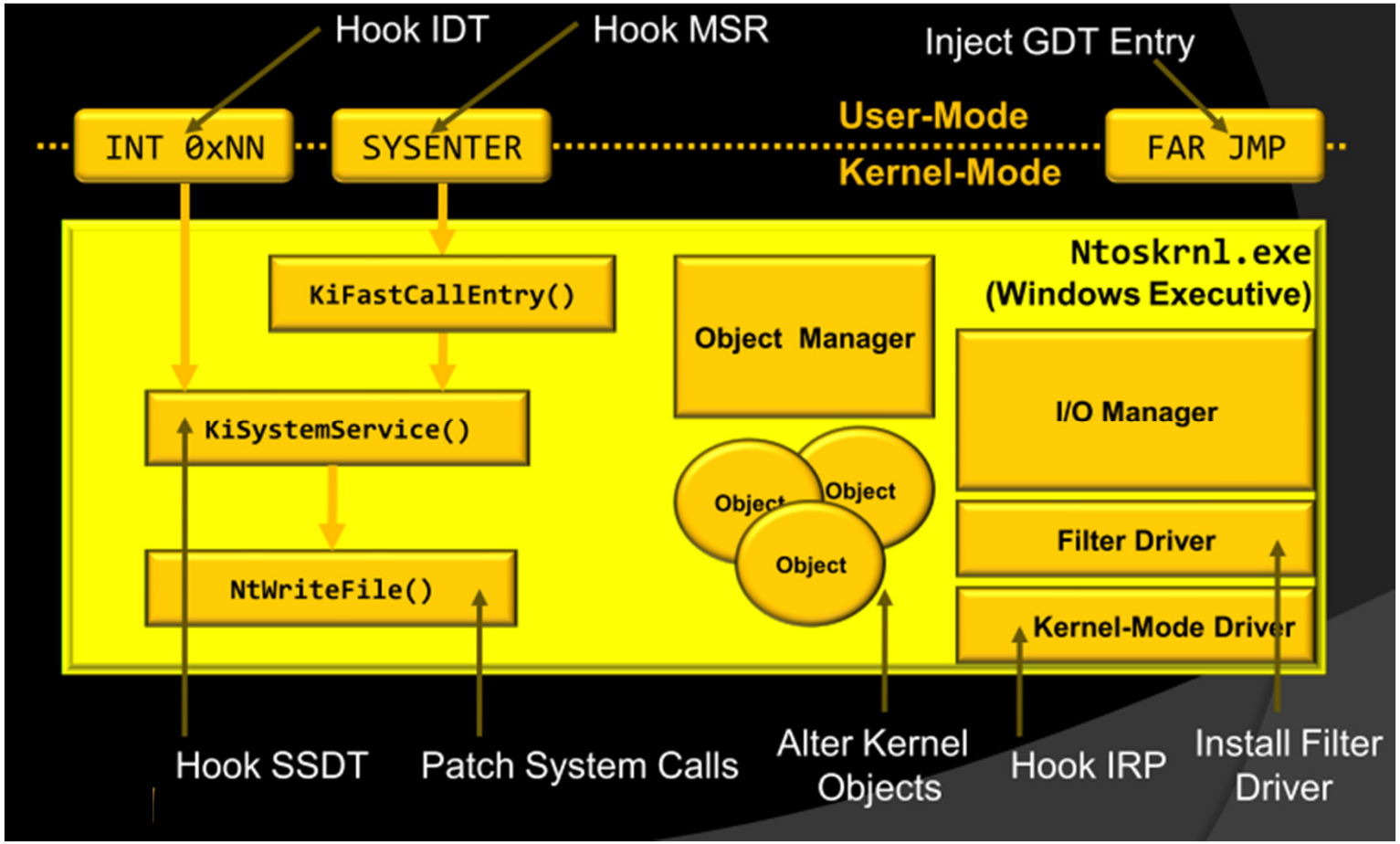
Source-Code Modification

- Người lập trình có thể chèn vào mã nguồn các dòng mã độc.
- Một back door được thực hiện như một bug trong phần mềm. Ví dụ một chương trình bị cài một lỗ hổng buffer overflow.
- Khó phát hiện vì xem như một bug

Sửa đổi trong User Mode



Sửa đổi trong Kernel Mode



Các thủ thuật thông thường

Mục đích sau cùng là đặt vào bộ nhớ code hay data

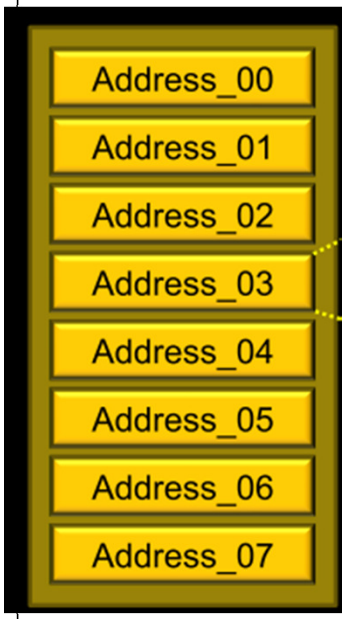
Sửa Data
- Call Tables
- Kernel Objects

Sửa Code có sẵn
- In-Place Patching
- Detour Patching

Đệ trình Code mới
- Filter Drivers
- Hypervisors
- DLL & Thread Injection
- COM & BHO Objects

Call Tables

- Là mảng của các con trỏ hàm (function pointers)



User-Mode Call Table	Shorthand	Description
Import Address Table	IAT	Specifies DLL imports

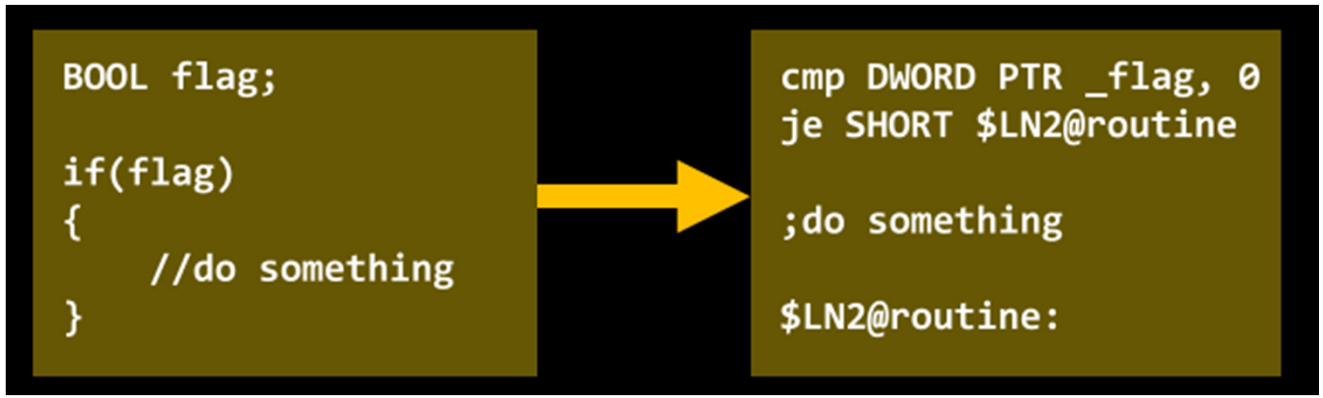
Kernel-Mode Call Tables (MSFT)	Shorthand	Description
System Service Descriptor Table	SSDT	Stores system call addresses
IRP Dispatch Table		Driver-specific, routes IRPs

Kernel-Mode Call Tables (Intel)	Shorthand	Description
Global Descriptor Table	GDT	System-wide, install call gates
Local Descriptor Table	LDT	Task-Specific, install call gates
Interrupt Descriptor Table	IDT	Stores interrupt handlers
Machine Specific Registers	MSRs	Used by `SYSENTER`

Kernel Objects

- Không phải đối tượng như trong lập trình hướng đối tượng
- Là sự trừu tượng hóa tài nguyên hệ thống
- Được hiện thực như một cấu trúc trong C
- Ví dụ: nt!_EPROCESS, nt!_DRIVER_OBJECT, nt!_TOKEN
- Có thể kiểm tra bằng kernel debugger

In-Place Patching



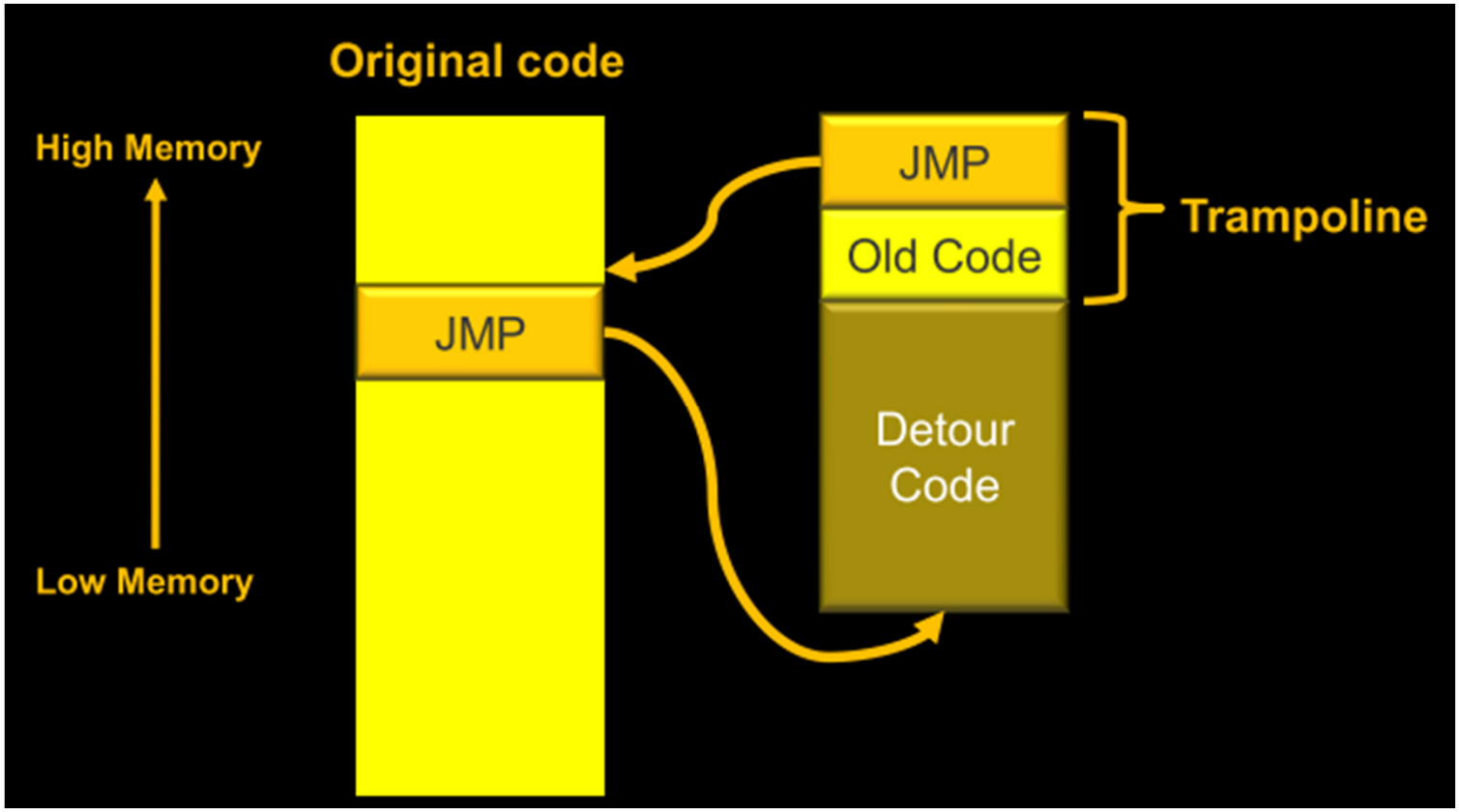
Có thể thay đổi code mà không phải thay đổi hướng thực thi

Thay je SHORT $LN2@routine (đó là 0x74 0x24)

bởi NOP NOP (đó là 0x90 0x90)

Code ở trong ngoặc luôn được thực thi!

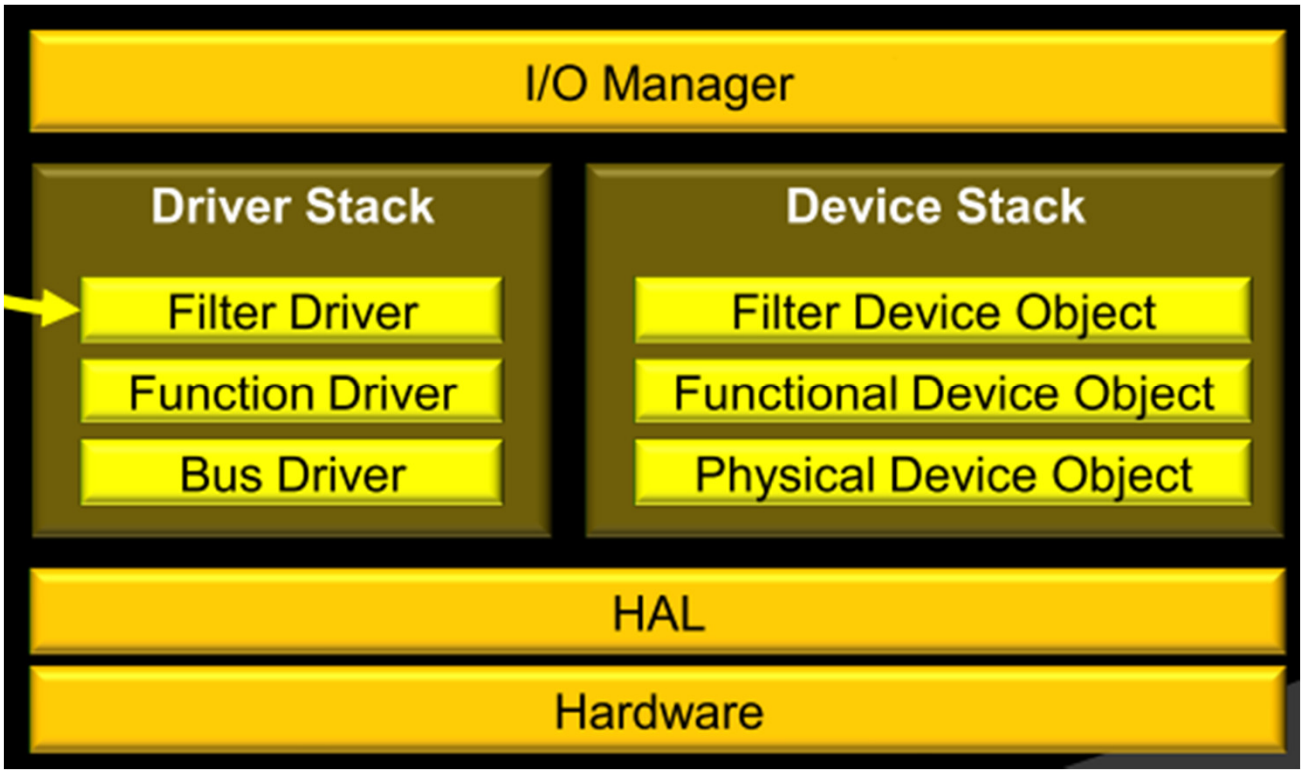
Detour Patching



Trampoline là tấm bạt che giấu

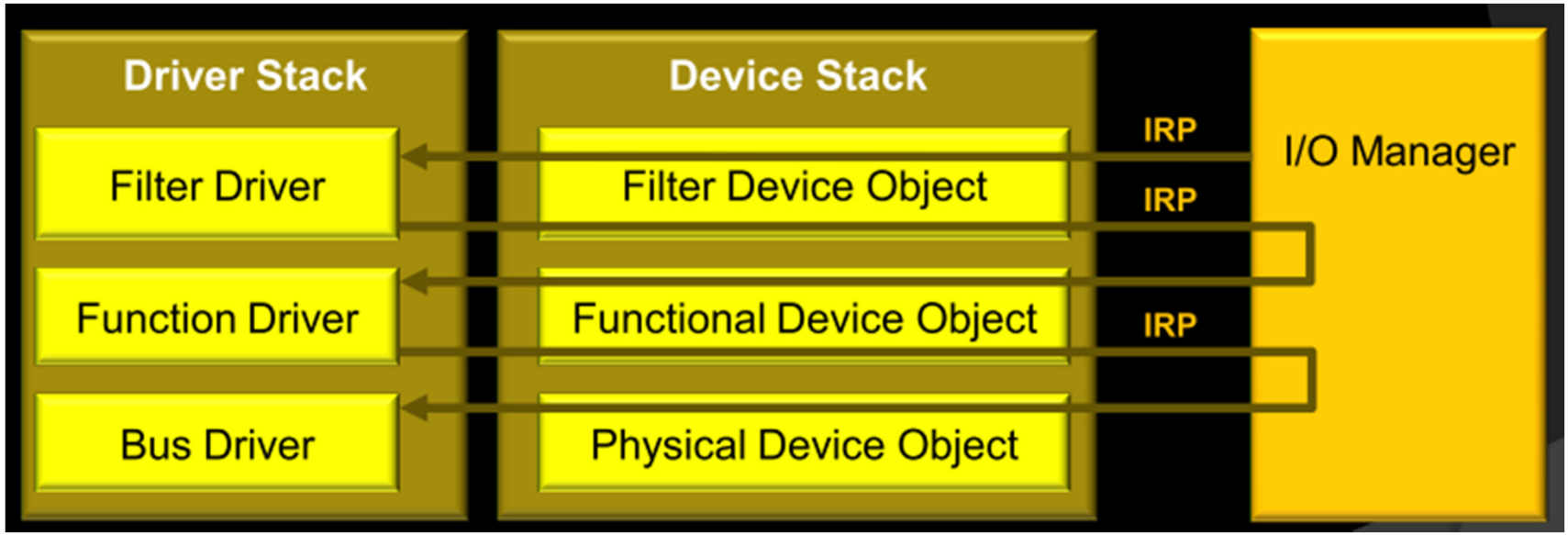
Filter Drivers (1/2)

- Filter driver được chèn vào một stack hiện hữu



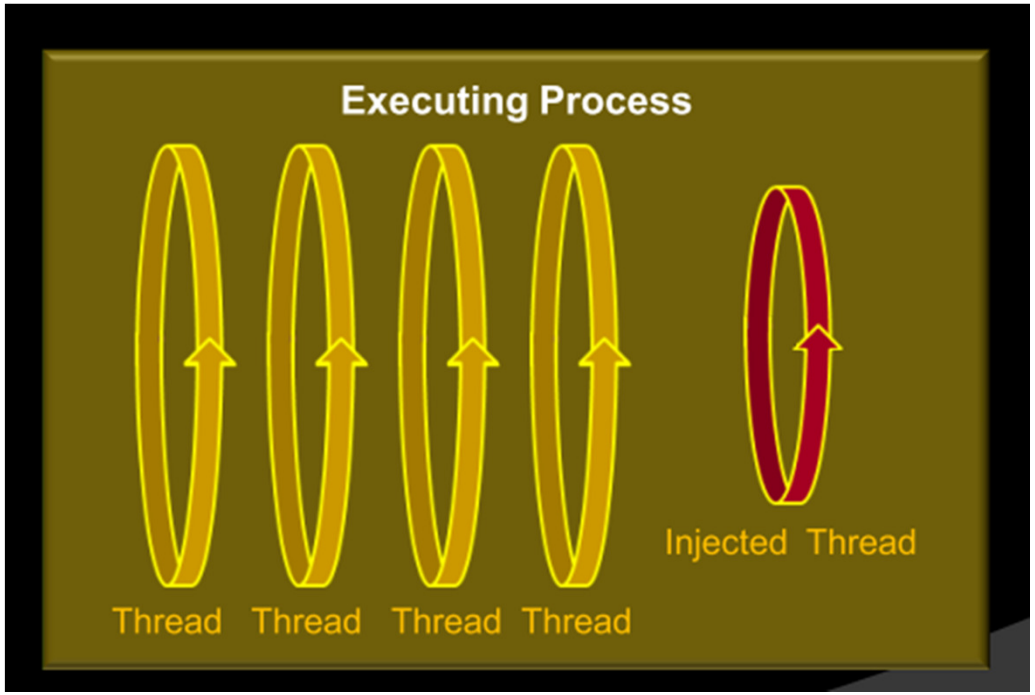
Filter Drivers (2/2)

- Can thiệp và sửa đổi các IRP (I/O Request Packet) đi qua chúng



DLL & Thread Injection

- Cách hiệu quả, Microsoft đã cài đặt các chức năng chống lại như UIPI*



Những gì không phải là rootkit

- Rootkit không phải là Exploit
- Rootkit không phải là Virus

Virus Problem

Các virus áp dụng công nghệ rootkit, do đó càng khó phát hiện và ngăn chặn

Một số giải pháp chống rootkit

Mod Tactic	Detection Countermeasures
Hooking	Manually parse memory, re-build tables
Object Patching	Cross-view detection, RAM Acquisition
Code Patching	Code is static → digital signature tests
Filter Drivers	Careful documentation, tools like `DeviceTree.exe`
Hypervisor	Time Analysis, TLB Profiling, etc.
Injected Code	Standard Auditing Tools, RAM Acquisition
Bootkit	Offline checksum of boot code

Kernel bị xâm hại như thế nào?

Các thành phần trọng yếu của Kernel

- Process management
- File access
- Security
- Memory management

Process management

- Các process cần CPU time→Kernel có code làm công việc cấp phát CPU time.
- Nếu OS có dùng threads, kernel sẽ lập lịch cho mỗi thread.
- Cấu trúc dữ liệu trong bộ nhớ theo dõi tất cả các thread và process→bằng cách sửa đổi các cấu trúc dữ liệu này, attacker có thể ẩn một process.

File access

- Các device driver được nạp để kiểm soát các hệ thống file khác nhau, như FAT32, NTFS.
- Kernel cung cấp một giao tiếp nhất quán cho các hệ thống file→bằng cách thay đổi code trong phần này của kernel, attacker có thể giấu các file hay thư mục một cách dễ dàng.

Security

- Kernel chịu trách nhiệm sau cùng trong việc áp đặt các qui tắc giữa các processes
- Trên UNIX và MS-Windows, kernel áp đặt quyền và dải bộ nhớ cho mỗi process →chỉ một vài sửa đổi code trong phần này có thể vô hiệu tất cả các cơ chế an ninh này.

Memory management

- Trong một số nền tảng phần cứng, một địa chỉ bộ nhớ có thể được ánh xạ sang nhiều vị trí vật lý (hai process được ánh xạ khác nhau).
- Khai thác cách thức làm việc này có thể 3 rất hữu dụng để ẩn data đối với các trình debug hay phần mềm truy tìm chứng cứ (forensic software).

Tiếp tục sinh tồn với Reboot

- Rootkit driver phải được nạp khi system boot.
- Cũng có nhiều thành phần phần mềm được nạp khi system boot
- Miễn là rootkit được gắn với một trong các sự kiện trong quá trình boot được liệt kê trong **7 slide kế tiếp**, nó cũng sẽ được nạp.

Dùng run key

- Run key (và các biến thể của nó) có thể dùng để nạp bất kỳ chương trình nào vào lúc boot.
- Chương trình được nạp có thể giải nén và nạp rootkit.
- Rootkit có thể ẩn run-key value một khi nó đã được nạp để không bị phát hiện.

Dùng một Trojan hay file bị nhiễm

- Bất kỳ tập tin .sys hay executable nào được nạp vào lúc boot đều có thể bị thay thế, hay loader code có thể được chèn vào theo cách tương tự như virus nhiễm vào một file.
- Một trojan DLL có thể được chèn vào search path
- Một DLL hiện hữu có thể bị thay thế một cách đơn giản.

Dùng các tập tin .ini

- Các tập tin .ini có thể bị thay đổi khiến cho các chương trình được chạy.
- Nhiều chương trình có các tập tin khởi động có thể chạy các lệnh và chỉ ra các tập tin DLL để nạp.

Đăng ký như một driver

- Rootkit có thể tự đăng ký như một driver được phép nạp vào lúc boot máy.
- Điều này yêu cầu tạo một registry key, tuy nhiên key này sẽ ẩn một khi rootkit đã được nạp.

Đăng ký như một add-on của một ứng dụng hiện hữu

- Để bổ sung một chức năng mở rộng vào các trình duyệt web
- Chức năng mở rộng được nạp khi ứng dụng nạp
- Phương pháp này khá hiệu quả để nạp rootkit.

Sửa đổi kernel ngay trên đĩa

- Kernel có thể bị sửa đổi một cách trực tiếp và được lưu trên đĩa
- Phải thay đổi một vài thứ trong boot loader sao cho kernel sẽ bỏ qua thủ tục kiểm tra tính toàn vẹn theo checksum (checksum integrity check)
- Kernel sẽ bị thay đổi vĩnh viễn và không có dirver nào phải đăng ký.

Sửa đổi boot loader

- Boot loader có thể bị sửa đổi để áp đặt các bản vá vào kernek trước khi nạp.
- Ưu điểm là kernel file không có biểu hiện bị thay đổi nếu hệ thống được phân tích offline.
- Việc sửa đổi boot-loader hoàn toàn có thể bị phát hiện bởi các công cụ thích hợp.

Đệ trình code vào Kernel

- Một cách đệ trình code vào kernel là dùng một module có thể nạp, được gọi là device driver hay kernel driver.
- Hầu hết các hệ điều hành hiện đại đều cho phép các chức năng mở rộng của kernel được nạp (hỗ trợ cho các nhà chế tạo phần cứng thứ ba)
- Bất kỳ code nào đều có thể được đệ trình thông qua một driver (không phải chỉ cho thiết bị ngoại vi)→full access to computer

Module tiêu biểu

- Linux-loadable module

```
int init_module(void)
{
}
void cleanup_module(void)
{
}
```

- Trong Windows device driver, entry point phải đăng ký function callbacks

```
NTSTATUS DriverEntry( ... )
{
    theDriver->DriverUnload = MyCleanupRoutine;
}
NTSTATUS MyCleanupRoutine()
{
}
```

Xây dựng Windows Device Driver

Device Driver đơn giản

```
#include "ntddk.h"
NTSTATUS DriverEntry( IN PDRIVER_OBJECT
theDriverObject,
IN PUNICODE_STRING theRegistryPath )'
{
   DbgPrint("Hello World!");
   return STATUS_SUCCESS;
}
```

Bộ công cụ phát triển Device Driver

- Driver Development Kit (DDK)
- Windows 2003 DDK.
- Can build drivers for Windows XP, using this version of the DDK
- The DDK provides two different build environments: the checked and the free build environments

Unload Routine

- Khi tạo một driver, tham số theDriverObject được truyền vào hàm main của driver. Nó chỉ đến một cấu trúc dữ liệu chức các con trỏ hàm. Một trong các con trỏ này được gọi là "unload routine."
- Nếu unload routine được set thì driver có thể được unload khỏi bộ nhớ.
- Ngược lại, không thể unload driver khỏi bộ nhớ, cần phải boot lại.

Thủ tục set con trỏ unload routine

- Trước hết cần tạo một hàm unload function, sau đó cài đặt con trỏ unload:

```
VOID OnUnload( IN PDRIVER_OBJECT DriverObject )
{
    DbgPrint("OnUnload called\n");
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT
theDriverObject,
                IN PUNICODE_STRING theRegistryPath)
{
    DbgPrint("I loaded!");
    // Initialize the pointer to the unload function
    // in the DriverObject
    theDriverObject->DriverUnload  = OnUnload;
    return STATUS_SUCCESS;}
```

Loading và Unloading một Driver

- PnPUtil
- Thêm một gói driver vào kho driver
- Liệt kê các gói driver trong kho
- Xóa gói driver

HẾT