RAIDE: **R**ootkit **A**nalysis **Id**entification **E**limination

by

Jamie Butler & Peter Silberman

Who Are We?

- Peter Silberman
  - Undergraduate College Student
  - Independent Security Research
  - Author of FUTo, PAIMEIdiff
  - Contributor to http://www.openRCE.org (VISIT THE SITE)
- Jamie Butler
  - CTO of Komoku http://www.komoku.com/
    - Software attestation
    - Rootkit detection
  - Author of *Rootkits: Subverting the Windows Kernel*
  - Co-author of Shadow Walker proof-of-concept memory subversion rootkit
  - Pioneer of Direct Kernel Object Manipulation (DKOM)

Agenda

- Overview of Rootkits
  - Hooks
    - Import Address Table (IAT)
    - KeServiceDescriptorTable
      - Inline
      - Entry overwrite
    - I/O Request Packet (IRP)
    - Interrupt Descriptor Table
  - Advanced Process Hiding
  - Detecting Hidden Processes
- RAIDE
- Demo using RAIDE

What is a rootkit

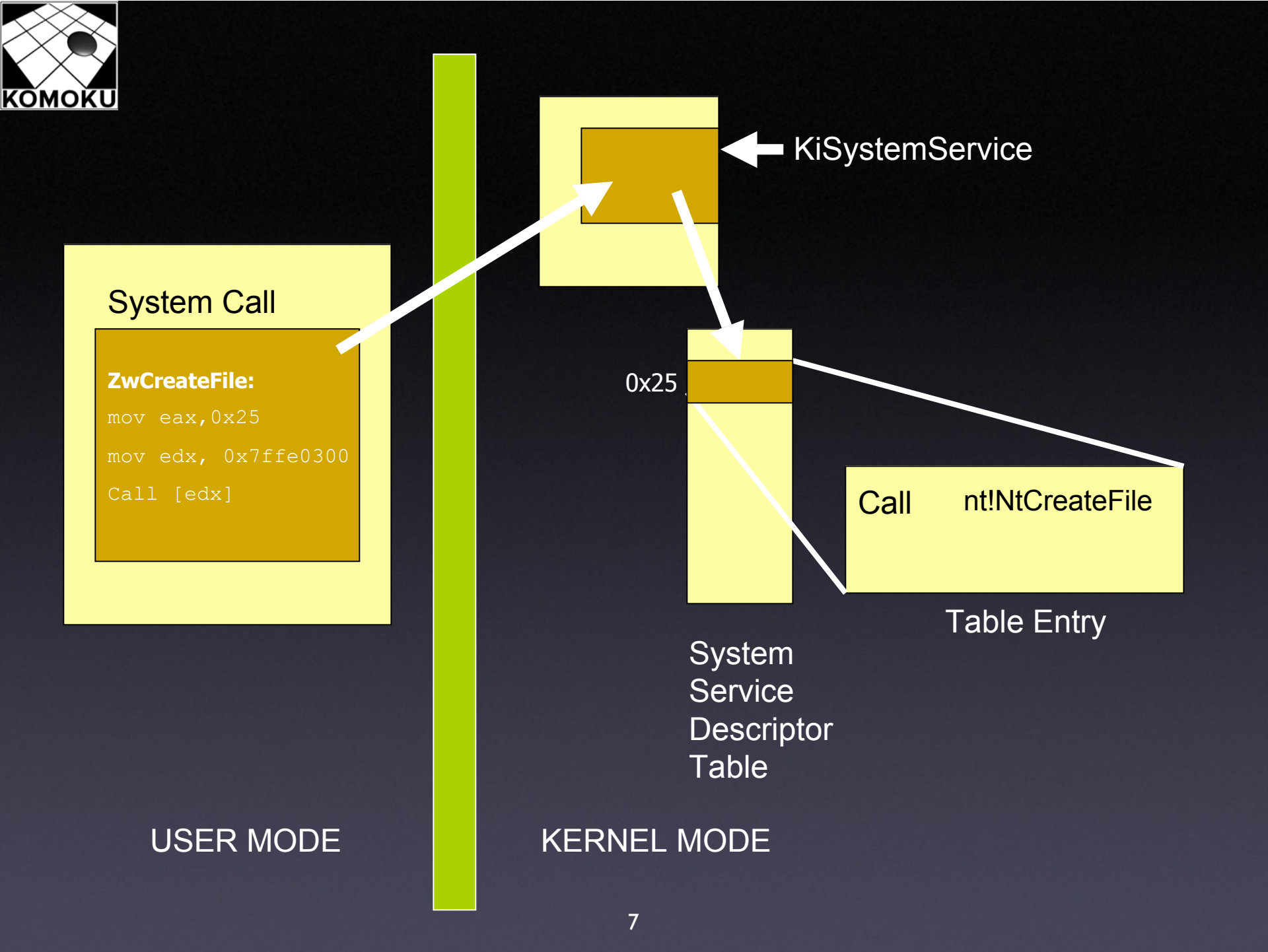
- Definition might include
  - a set of programs which patch and Trojan existing execution paths within the system
    - Hooks or Modifies existing execution paths of important operating system functions
  - The key point of a rootkit is stealth.
    - Rootkits that do not hide themselves are not then using stealth methods and will be visible to administrative or forensic tools

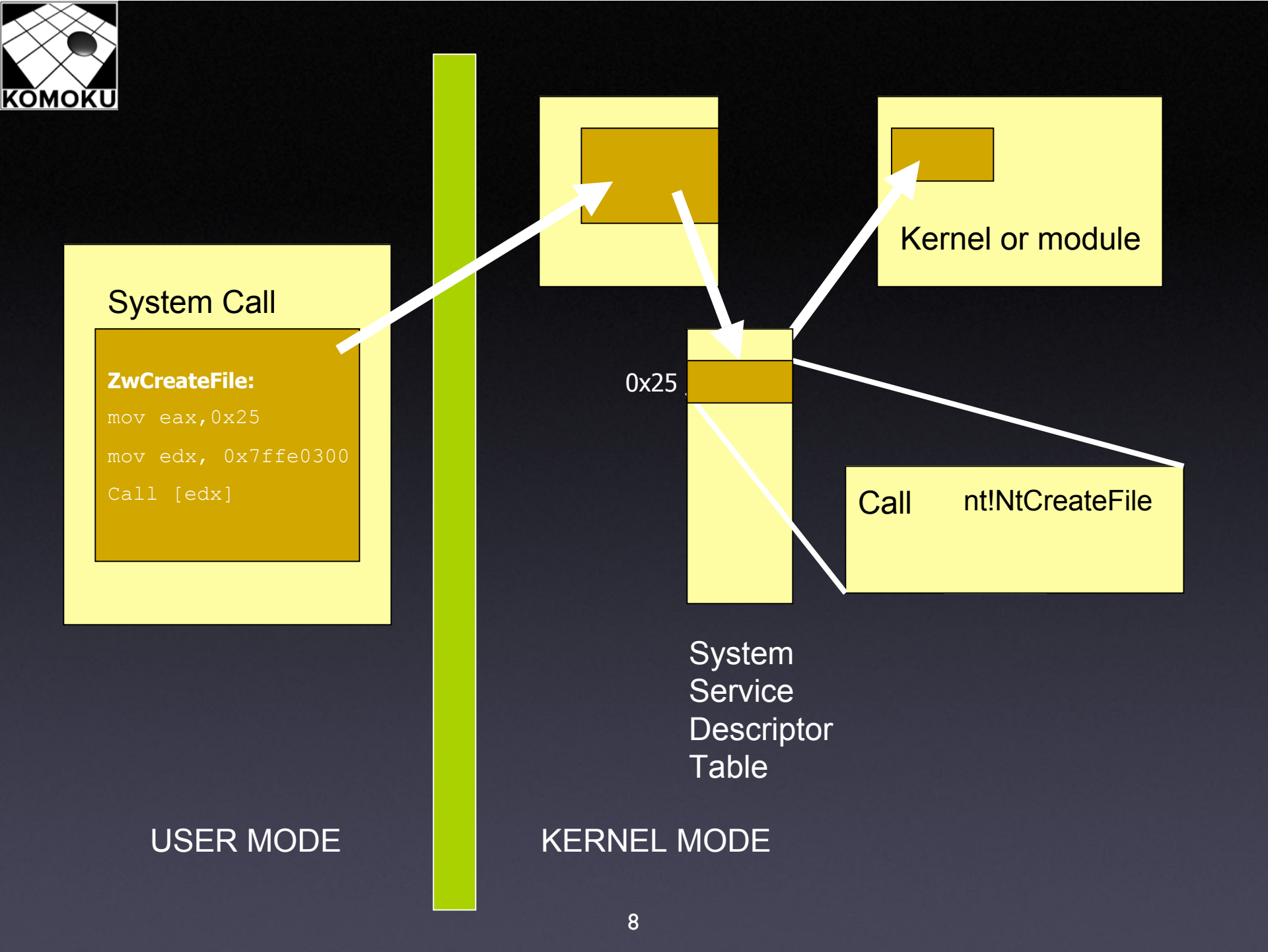
Hooking in User Land

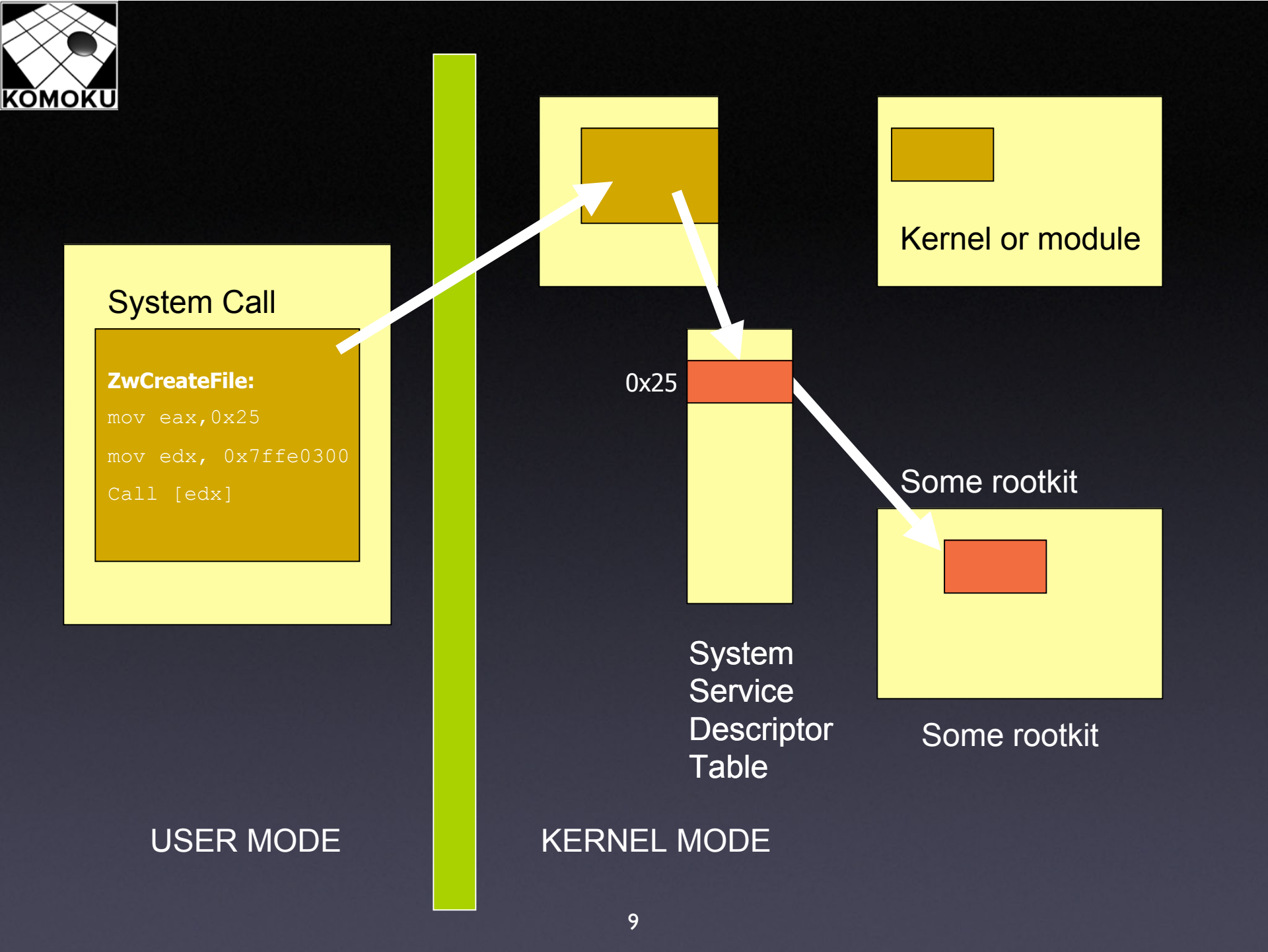
- IAT hooks
  - Hooking code must run in or alter the address space of the target process
    - If you try to patch a shared DLL such as KERNEL32.DLL or NTDLL.DLL, you will get a private copy of the DLL.
  - Three documented ways to gain execution in the target address space
    - CreateRemoteThread
    - Globally hooking Windows messages
    - Using the Registry
      - HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs

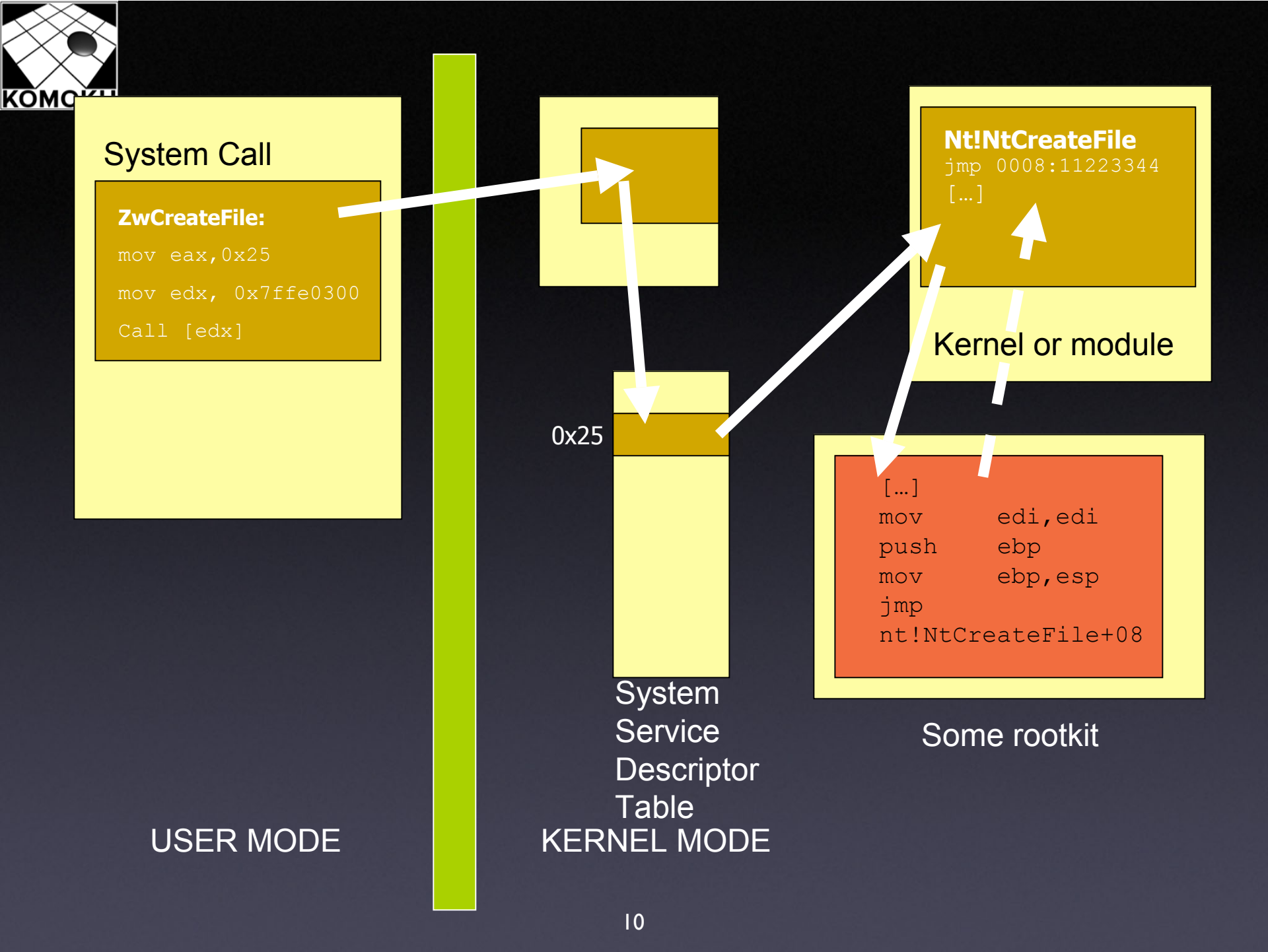
Hooking in Kernel Space

- The operating system is global memory
- Does not rely on process context
  - Except when portions of a driver are pageable
- By altering a single piece of code or a single pointer to code, the rootkit subverts every process on the system


KOMOKU
KiSystemService
System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
0x25
Call nt!NtCreateFile
Table Entry
System
Service
Descriptor
Table
USER MODE
KERNEL MODE


KOMOKU
System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
Kernel or module
0x25
Call nt!NtCreateFile
System
Service
Descriptor
Table
USER MODE
KERNEL MODE


KOMOKU
System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
Kernel or module
0x25
Some rootkit
System Service Descriptor Table
Some rootkit
USER MODE
KERNEL MODE


System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
0x25
System
Service
Descriptor
Table
Nt!NtCreateFile
jmp 0008:11223344
[…]
Kernel or module
[…]
mov edi,edi
push ebp
mov ebp,esp
jmp
nt!NtCreateFile+08
Some rootkit
USER MODE
KERNEL MODE

I/O Manager and IRP Hooking

- System calls used to send commands
  - NtDeviceIoControlFile
  - NtWriteFile
  - Etc.
- Requests are converted to I/O Request Packets (IRPs)
- IRPs are delivered to lower level drivers

I/O Manager and IRP Hooking

- Every driver is represented by a DRIVER_OBJECT
- IRPs are handled by a set of 28 function pointers within the DRIVER_OBJECT
- A rootkit can hook one of these function pointers to gain control

Interrupt Descriptor Table Hooks

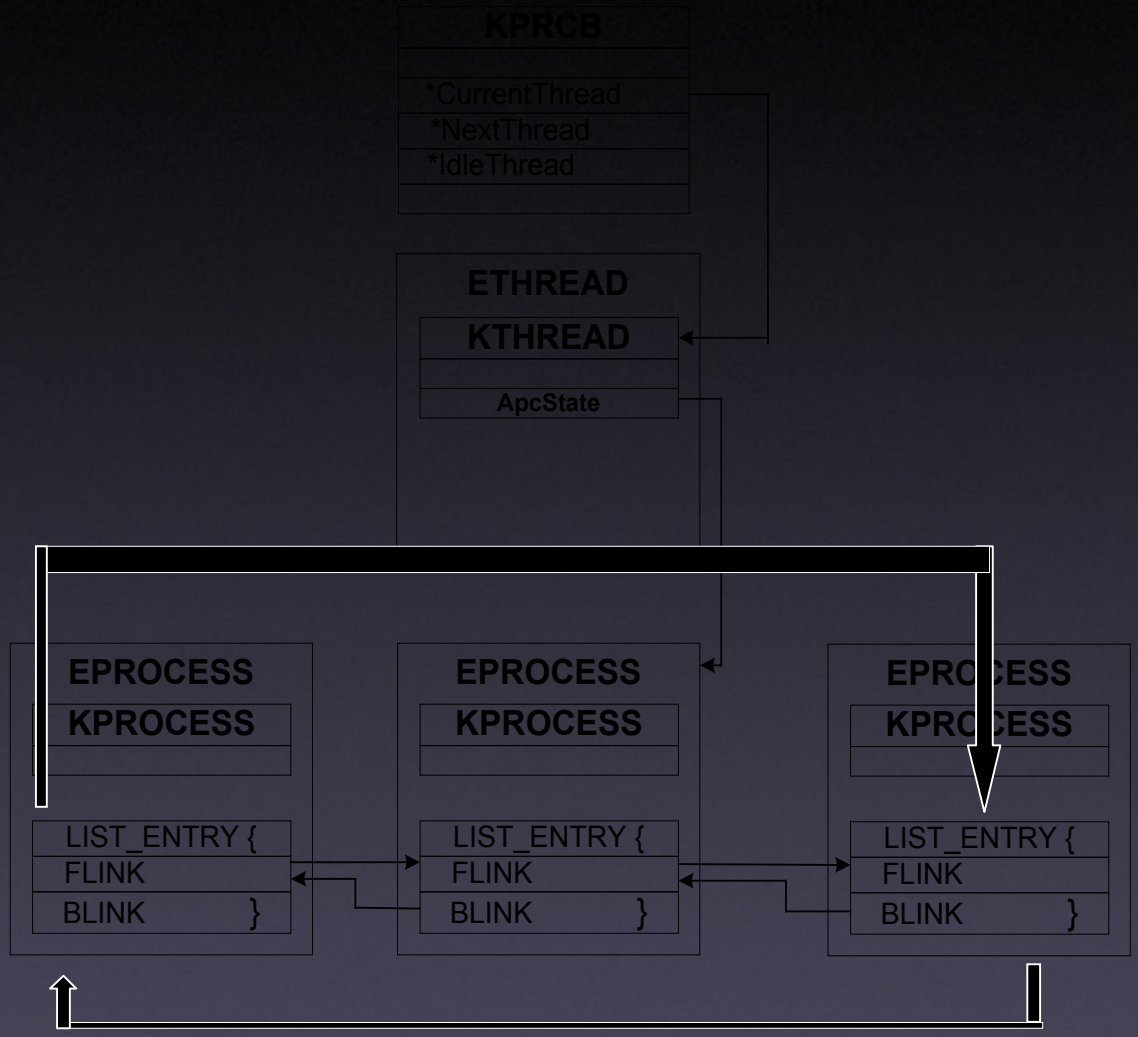
- Each CPU has an IDT
- IDT contains pointers to Interrupt Service Routines (ISRs)
- Uses for IDT hooks
  - Take over the virtual memory manager
  - Single step the processor
  - Intercept keystrokes

Advanced Process Hiding

Hiding Processes - Review

- DKOM Uses
  - To hide a process
    - Locate the EPROCESS block of the process to hide
    - Change the process behind it to point to the process after the process you are hiding
    - Change the process after it to point to the process before the one you are trying to hide
  - Add Privileges to Tokens
  - Add Groups to Tokens
  - Manipulate the Token to Fool the Windows Event Viewer
  - Hide Ports

Hiding Processes - Windows



FUTo – Hiding From the Tables

- FUTo
  - Uninformed Journal Vol. 3 (http://www.uninformed.org)
  - New version of FU hence the ‘To’
  - Hides from IceSword and Blacklight
    - Option –pngh bypasses as of (06/26/06):
      - Blacklight (F-Secure)
      - AntiRootkit (BitDefender)
- Removes itself from the PspCidTable

PspCidTable (PspPidTable)

- PspCidTable
  - Job of PspCidTable is to keep track of all the processes and threads
    - PspCidTable's indexes are the PIDs of processes.
    - Returns the address of the EPROCESS of a process at the location corresponding to the PID.
- Problems:
  - Relying on a single data structure is not a very robust
  - By altering one data structure much of the OS has no idea the hidden process exists

Kernel Structures: The Tables

- Handle Table:
  - Handles are an index into the Handle Table for a particular object
  - Objects represent processes, threads, tokens, events, ports, etc.
  - The Object Manager must do the translation from a handle to an object
  - The Object Manager consults the Security Reference Monitor to determine access to the object
  - Every process has its own handle table to keep track of the handles it owns

Kernel Structures: Handle Tables

```
lkd> dt nt!_HANDLE_TABLE
   +0x000 TableCode                : Uint4B
   +0x004 QuotaProcess             : Ptr32 _EPROCESS
   +0x008 UniqueProcessId          : Ptr32 Void
   +0x00c HandleTableLock          : [4] _EX_PUSH_LOCK
   +0x01c HandleTableList          : _LIST_ENTRY
   +0x024 HandleContentionEvent : _EX_PUSH_LOCK
   +0x028 DebugInfo                : Ptr32 _HANDLE_TRACE_DEBUG_INFO
   +0x02c ExtraInfoPages           : Int4B
   +0x030 FirstFree                : Uint4B
   +0x034 LastFree                 : Uint4B
   +0x038 NextHandleNeedingPool: Uint4B
   +0x03c HandleCount              : Int4B
   +0x040 Flags                    : Uint4B
   +0x040 StrictFIFO               : Pos 0, 1 Bit
```

Handle Table Translation

```
test.exe ProcessId 152
{
HANDLE hProcess;
hProcess = OpenProcess(PROCESS_ALL_ACCESS, 0,
132);
if(hProcess == INVALID_HANDLE)
                return 0;
TerminateProcess(hProcess);
}
```

hProcess = 0x03

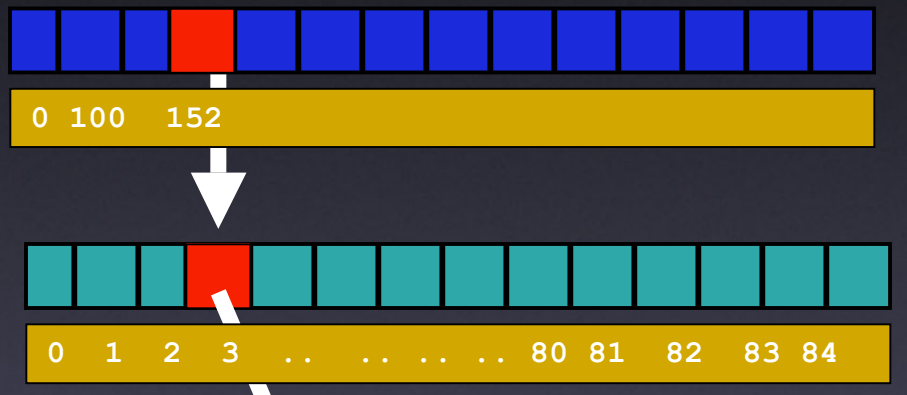
ZwTerminateProcess(**hProcess**);

NtTerminateProcess:
PVOID obj =
TranslateHandleToObject(**hProcess**);

TranslateHandleToObject
Process = PspCidTable[**PsGetCurrentProcessById()**];

if(Process == NULL) return 0;

return Process->ObjectTable[**hProcess**];



Object:

ObjectType = OBJ_PROCESS

Object = 0x8014231

Handle Table Translation

```
test.exe ProcessId 152
{
HANDLE hProcess;
hProcess = OpenProcess(PROCESS_ALL_ACCESS, 0,
132);
if(hProcess == INVALID_HANDLE)
                return 0;
TerminateProcess(hProcess);
}
```

hProcess = 0x03

ZwTerminateProcess(**hProcess**);

NtTerminateProcess:
PVOID obj =
TranslateHandleToObject(**hProcess**);

```
TranslateHandleToObject
Process = PspCidTable[ PsGetCurrentProcessById()];
if( Process == NULL) return 0;
return Process->ObjectTable[hProcess];
```

Object:
ObjectType = OBJ_PROCESS
Object = 0x8014231

Detecting Processes

- Blacklight Beta
  - Released in March 2005
  - Good hidden process and file detection
- IceSword 1.12
  - Robust tool offering:
    - SSDT Hook Detection
    - Hidden File and Registry Detection
    - Hidden Process Detection
    - Hidden Ports and socket communication Detection
- Common flaw
  - Both applications rely upon the PspCidTable for detection

Detecting Hidden Processes
PID Bruteforce

- Blacklight
  - Bruteforces PIDs 0x0 - 0x4E1C
    - Calls OpenThread on each PID
      - If Success store valid PID
    - Else Continue Loop
  - Finished looping, take list of known PIDs and compare it to list generated by calling CreateToolhelp32Snapshot
  - Any differences are hidden processes
    - Called Cross-View method or Difference Based Method

RAIDE

RAIDE

- What is RAIDE?
- What makes RAIDE different than Blacklight, RKDetector, Rootkit Revealer, VICE, SVV, SDTRestore, AntiRootkit?
- What doesn't RAIDE do?

What is RAIDE

- RAIDE is a complete toolkit offering:
  - Hidden Process Detection (Blacklight, AntiRootkit, Others)
  - Hook Detection (SDTRestore, SVV, VICE)
  - Hook Restoration (SDTRestore, SVV)
  - IDT Detection
    - Memory Subversion Detection
  - Hidden Process Features
    - Relink processes to make it visible
    - Close Hidden Processes
  - Method Detection
    - Hidden Process Method Detection – Example hook, DKOM, etc.
    - Hook Detection Method

What Makes RAIDE Different?

- RAIDE combines most existing tools
- RAIDE detects Memory Subversion
- RAIDE gives the user more information about hidden processes and Hooks
- RAIDE does **not** use IOCTL's to communicate
- RAIDE identifies NDIS hooks
- RAIDE can restore non-exported ntoskrnl functions

What Doesn't RAIDE Do?

- RAIDE does not detect hidden files, folders, or registry keys
- RAIDE does not restore Driver IRP hooks
- RAIDE does not restore IDT hooks (future maybe?)
- RAIDE does not prevent a rootkit from loading
- RAIDE is not a substitute for common sense

RAIDE Communication

- RAIDE communication designed to thwart Crappy And Stupid Application Specific Attacks (CASASA)
- RAIDE uses Shared Memory segments to pass information kernel land →→ user land
  - Shared Memory contains only encrypted data
  - Communication uses randomly named events for signaling
  - Uses randomly generated process names
    - RAIDE spawns a user process from a driver to do a Difference Based or Cross-View comparison
    - The spawned process looks like any other process spawned from userland.

Hidden Process Detection

- Goal for Process Detection:
  - Signature that can not be zeroed out
  - Signature that is unique
  - Signature must not have false positives

Hidden Process Detection

- Signature:
  - Locate pointers to “ServiceTable”
    - ServiceTable = nt!KeServiceDescriptorTableShadow
    - ServiceTable = nt!KeServiceDescriptorTable
  - Contained in all ETHREAD
- Hidden Process:
  - Spawn a process with random name
    - Spawned process generates process list
      - sends processes list visible to RAIDE
    - RAIDE compares the two lists finding the differences
      - hidden processes

Hidden Process Method Detection

- To detect hidden process methods, we need to know the two methods most commonly used.
  - DKOM
  - PspCidTable
- If the process is not visible by walking ActiveProcessList in the EPROCESS block then it was hidden using the DKOM method.
  - However for it to be hidden with the DKOM method it has to be visible in the PspCidTable, so RAIDE will walk that as well.
  - If it is hidden in both it uses the FUTo method.

Shadow Walker Detection: Illuminating the Shadows

- Shadow Walker relies on IDT hook
  - Check IDT 0x0e for a hook
    - SW could modify itself to hide the IDT hook with an inline hook
- Other detection schemes out there
  - Remapping Memory
    - By remapping, we mean remapping a given physical frame to a new virtual address (i.e. like the shared memory concept).

Forensics

- Hook Restoration
- Relinking Processes
- Dumping Processes

Hook Restoration

- If an SSDT entry overwrite hook is detected
  - Open ntoskrnl
  - Obtain KeServiceDescriptorTable from file on disk
  - Obtain original address for hooked index
  - Recalculate address
  - “re-hook” SSDT index with original address

Hook Restoration

- If it is an inline hook:
  - Open ntoskrnl on disk
  - Obtain original function address
  - Read first few instructions
  - Restore first few instructions
    - Can restore as many instructions as needed

Relinking Processes

- DKOM is common hiding method
  - DKOM relies on unlinking the EPROCESS link pointers
  - Restore link pointers by passing the System EPROCESS and the hidden EPROCESS to *InsertTailList*
  - Allows user to see process

Dumping Process

- Dumping Process
  - Allows Security Analysts to reverse the executable or system file and see what it was doing.
  - Does not matter if the file is originally hidden on the HD.
    - Dump file is renamed and put in the working directory.
    - Dumping lets analysts bypass any packer protection.

Thanks

- Peter: bugcheck, xbud, thief, skape, pedram, greg h, nologin/research'ers, f-secure labs.
- Jamie: Lil' L, lonerancher, Barns, Greg, and Bugcheck

DEMO

Questions?