nsa Network Security and Architecture Laboratory

DEFCON THIRTEEN

Georgia Institute of Technology

Surgical Recovery from Kernel-Level Rootkit Installations

Speaker: Julian Grizzard

July 2005

Latest Slides and Tools

PLEASE DOWNLOAD THE LATEST SLIDES AND TOOLS

[Latest slides available]

http://www.ece.gatech.edu/research/labs/nsa/presentations/dc13_grizzard.pdf

[Latest system call table tools]

http://www.ece.gatech.edu/research/labs/nsa/sct_tools.shtml

[Latest spine architecture work]

http://www.ece.gatech.edu/research/labs/nsa/spine.shtml

Talk Overview

Talk focuses on Linux i386 based systems

- **Rootkit background**
- System call table tools
  - Demos
- L4 microkernel introduction
- Spine architecture
- Intrusion recovery system (IRS)
  - Demos
- Concluding remarks

Rootkit Functionality

- Retain Access
  - Trojan sshd client with hard coded user/pass for root access
  - Initiate remote entry by specially crafted packet stream
- Hide Activity
  - Hide a process including resource usage of process
  - Hide malicious rootkit kernel modules from lsmod

Additional Malware Functionality

- Information harvesting
  - Credit cards
  - Bank accounts
- Resource usage
  - Spam relaying
  - Distributed denial of service

User-Level versus Kernel-Level

- User-Level
  - Modify/replace system binaries
  - e.g. *ps, netstat, ls, top, passwd*
- Kernel-Level
  - Modify/replace kernel process
  - e.g. system call table

History of Kernel-Level Rootkits

- Heroin – October 1997
  - First public LKM
- Knark – June 1999
  - Highly popular LKM
- SucKIT – December 2001
  - First public /dev/kmem entry
- Adore-ng 0.31 – January 2004
  - Uses VFS redirection; works on Linux 2.6.X

Kernel-Level Rootkit Targets

- System call table
- Interrupt descriptor table
- Virtual file system layer
- Kernel data structures

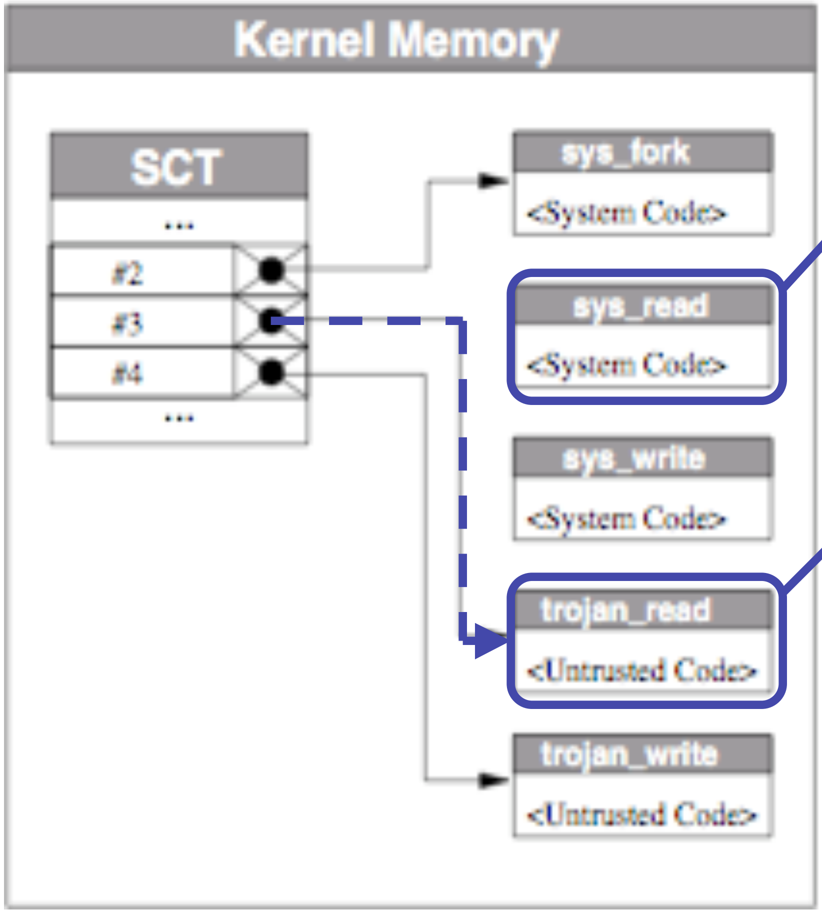
Kernel Entry

- Linux kernel module (LKM)
- /dev/kmem, /dev/mem, /dev/port
- Direct memory access (DMA)
- Modify kernel image on disk

System Call Table Modifications

- System calls are the main gateway from user space to kernel space
- Most commonly targeted kernel structure
- Can redirect individual system calls or the entire table

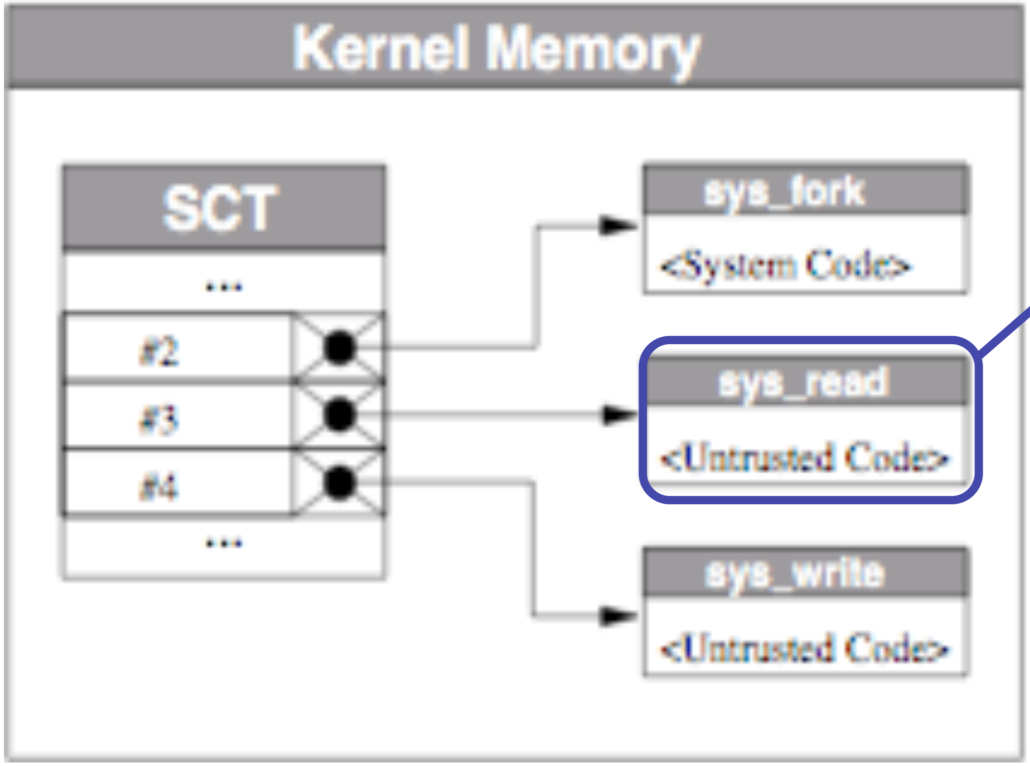
Entry Redirection



Original *read* system call. No longer pointed to by SCT.

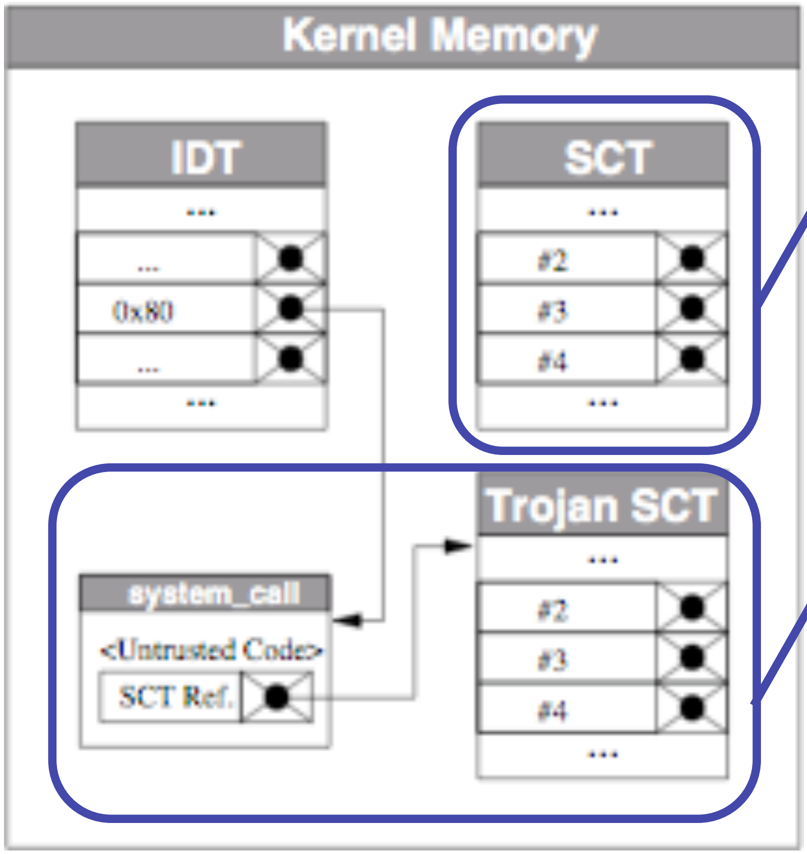
Trojaned *read* system call. Active SCT points to it.

Entry Overwrite



System call code overwritten; SCT still intact

Table Redirection



- Original SCT intact
- Original system calls intact

- Handler points to Trojan table

/dev/kmem Details from SucKIT

- SucKIT accesses kernel memory from user space
- Redirects entire system call table
- How does sucKIT find the system call table?
- How does sucKIT allocate kernel memory?

Find System Call Handler

```
struct idtr idtr;
struct idt idt80;
ulong old80;

/* Pop IDTR register from CPU */
asm("sidt %0" : "=m" (idtr));

/* Read kernel memory through /dev/kmem */
rkm(fd, &idt80, sizeof(idt80), idtr.base +
0x80 * sizeof(idt80));

/* Compute absolute offset of
 * system call handler for kmem
 */
old80 = idt80.off1 | (idt80.off2 << 16);
```

Kmalloc as a System Call (sucKIT)

```
#define rr(n, x) ,n ((ulong) x)
#define __NR_oldolduname 59
#define OURSYS __NR_oldolduname
#define syscall2(__type, __name, __t1, __t2)    \
   __type __name(__t1 __a1, __t2 __a2)          \
   {                                            \
      ulong __res;                              \
      __asm__ volatile                          \
      ("int $0x80"                              \
      : "=a" (__res)                            \
      : "0" (__NR_##__name)                     \
      rr("b", __a1)                             \
      rr("c", __a2));                           \
      return (__type) __res;                    \
   }
#define __NR_KMALLOC OURSYS
static inline syscall2(ulong, KMALLOC, ulong, ulong);
```

Example Kernel-Level Rootkits

Rootkit	Kernel Entry	Modification
heroin	Module	SCT Entry Redirection
knark	Module	SCT Entry Redirection
adore	Module	SCT Entry Redirection
sucKIT	User	SCT Table Redirection
zk	User	SCT Table Redirection
r.tgz	User	SCT Table Redirection
adore-ng	Module	VFS Redirection

Talk Overview

- Rootkit background
- **System call table tools**
  - **Demos**
- L4 microkernel introduction
- Spine architecture
- Intrusion recovery system (IRS)
  - Demos
- Concluding remarks

System Call Table Tools

- Developed tools that can query the state of the system call table and repair it
- Tools based on sucKIT source code and work from user space
- Algorithm to recover from rootkits is similar to algorithm used by rootkits

Algorithm (x86 architecture)

1) Copy clean system calls to kernel memory
2) Create new system call table
3) Copy system call handler to kmem
4) Query the *idtr* register (interrupt table)
5) Set *0x80*ith entry to new handler

Details

- Use a known good kernel image and rip out the system call table with gdb
- Address of system call table must be set in system call handler

Copying Kernel Functions

- Some trickery involved with algorithm
- x86 code has *call* instructions with a relative offset parameter
- Could recompile the code
- We chose to recompute relative offset and modify the machine code

Demos

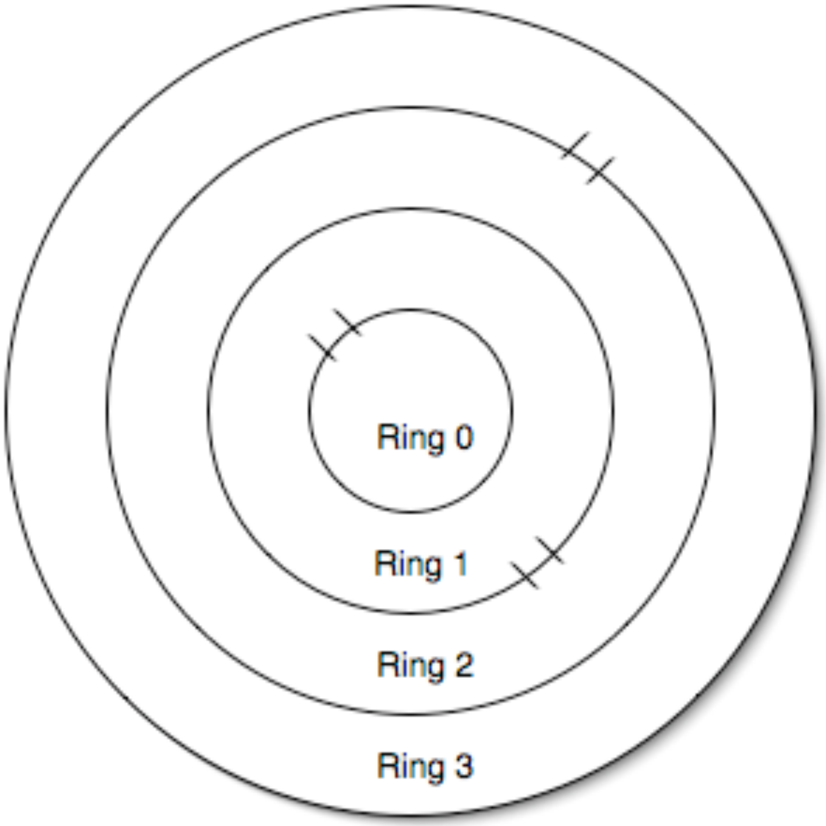
System Call Table Tools Demonstration

Talk Overview

- Rootkit background
- System call table tools
  - Demos
- **L4 microkernel introduction**
- Spine architecture
- Intrusion recovery system (IRS)
  - Demos
- Concluding remarks

Intel Descriptor Privilege Level

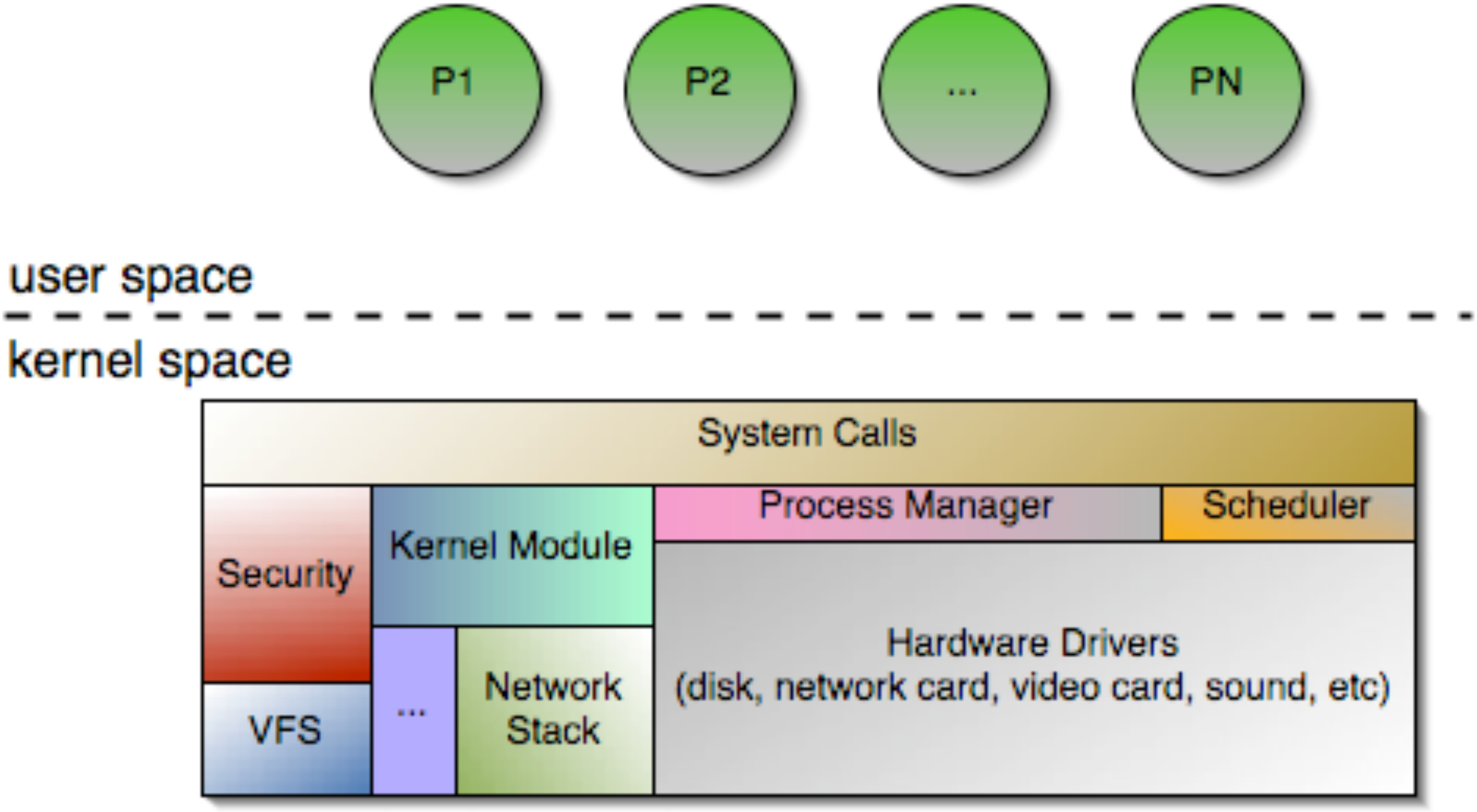
- Level 3
  - Minimal hardware access
  - User space processes run at level 3
- Level 2
  - Limited hardware access
  - N/A in Linux
- Level 1
  - Limited hardware access
  - N/A in Linux
- Level 0
  - Unlimited hardware access
  - Kernel space threads run at level 0



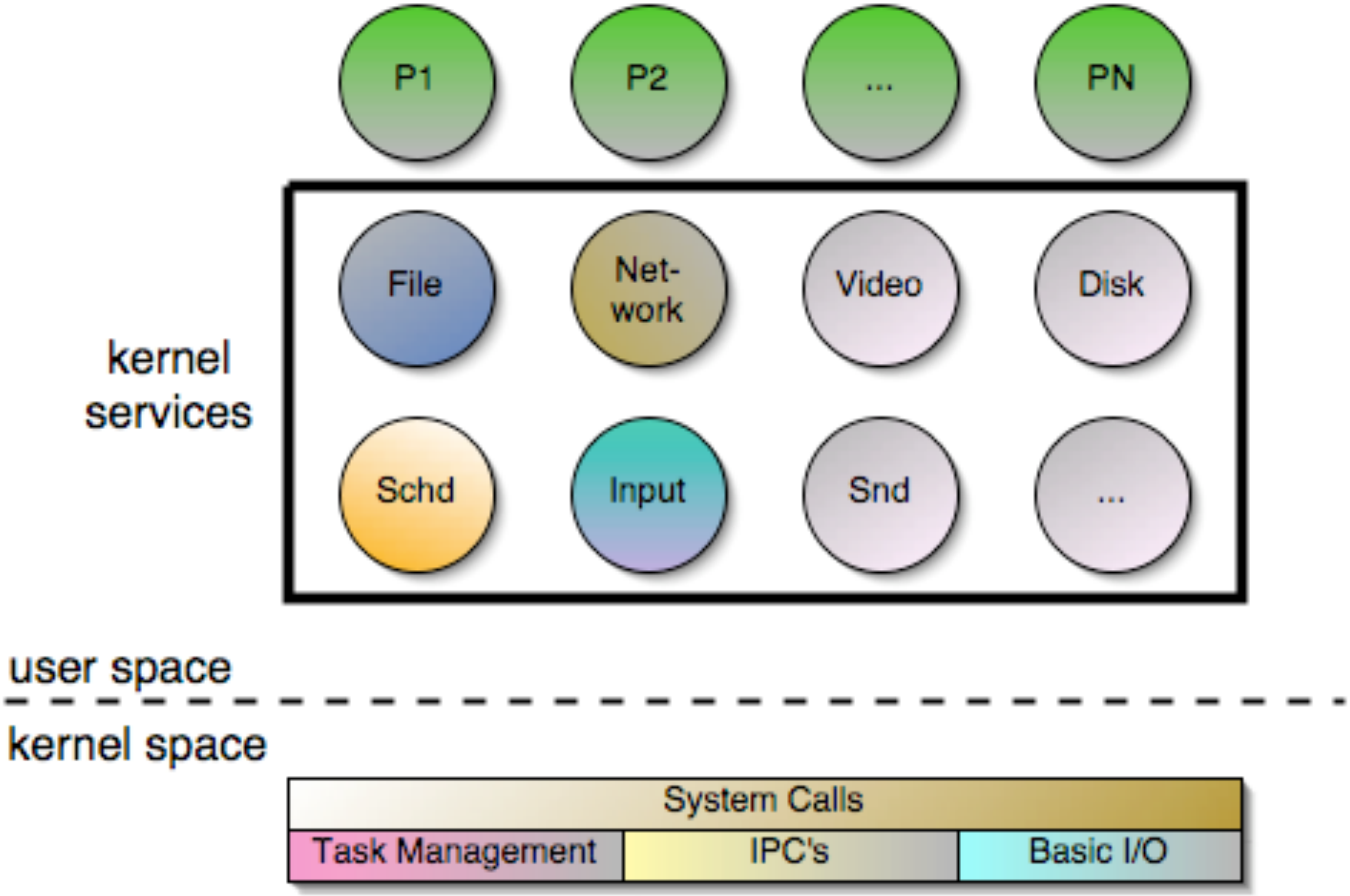
Virtual Machines/Hypervisors

- VMware
- User Mode Linux
- Xen
- L4

Monolithic Operating System



Microkernel Operating System



History of Microkernels

- Mach project started at CMU (1985)
- QNX
- Windows NT
- LynxOS
- Chorus
- Mac OS X

Microkernel Requirements

- Tasks
- IPC
- I/O Support

That’s it!

L4 System Calls (Fiasco)

- **9 IPC Calls**
  - l4_ipc_call, l4_ipc_receive
  - l4_ipc_reply_and_wait
  - l4_ipc_send_deceiting, l4_ipc_reply_deciting_and_wait
  - l4_ipc_send, l4_ipc_wait
  - l4_nchief
  - l4_fpage_unmap
- **5 Thread calls**
  - l4_myself
  - l4_task_new
  - l4_thread_ex_regs
  - l4_thread_schedule
  - l4_thread_switch

L4 IPC’s

- Fast IPCS
- Flexpages
- Clans and chiefs
- System calls, page faults are IPC’s

L4 I/O (from Fiasco lecture slides)

- Hardware interrupts: mapped to IPC
  - Special thread id for interrupts
  - IPC sender indicates interrupt source
  - Kernel provides no sharing support, one thread per interrupt
  - Malicious driver could potentially block all interrupts if given access to PIC
  - Cli/sti only allowed in kernel and trusted servers
- I/O memory and I/O ports: flexpages
- Missing kernel feature: pass interrupt association
  - Security hole
- I/O port access
- DMA - big security risk

Rmgr (lecture slides)

- Resources --- serves page faults
  - Physical memory
  - I/O ports
  - Tasks
  - Interrupts

Booting the System (lecture slides)

- Modified grub
- Multi-boot specification
- Rmgr, sigma0, root task (rmgr II), …
- IDT
  - General Protection Exception #13
  - Page Fault #14
  - Divide by zero #0
  - Invalid opcode #6
  - System calls Int30 IPC
- Global Descriptor Table (GDT) vs. Local Descriptor Table (LDT)

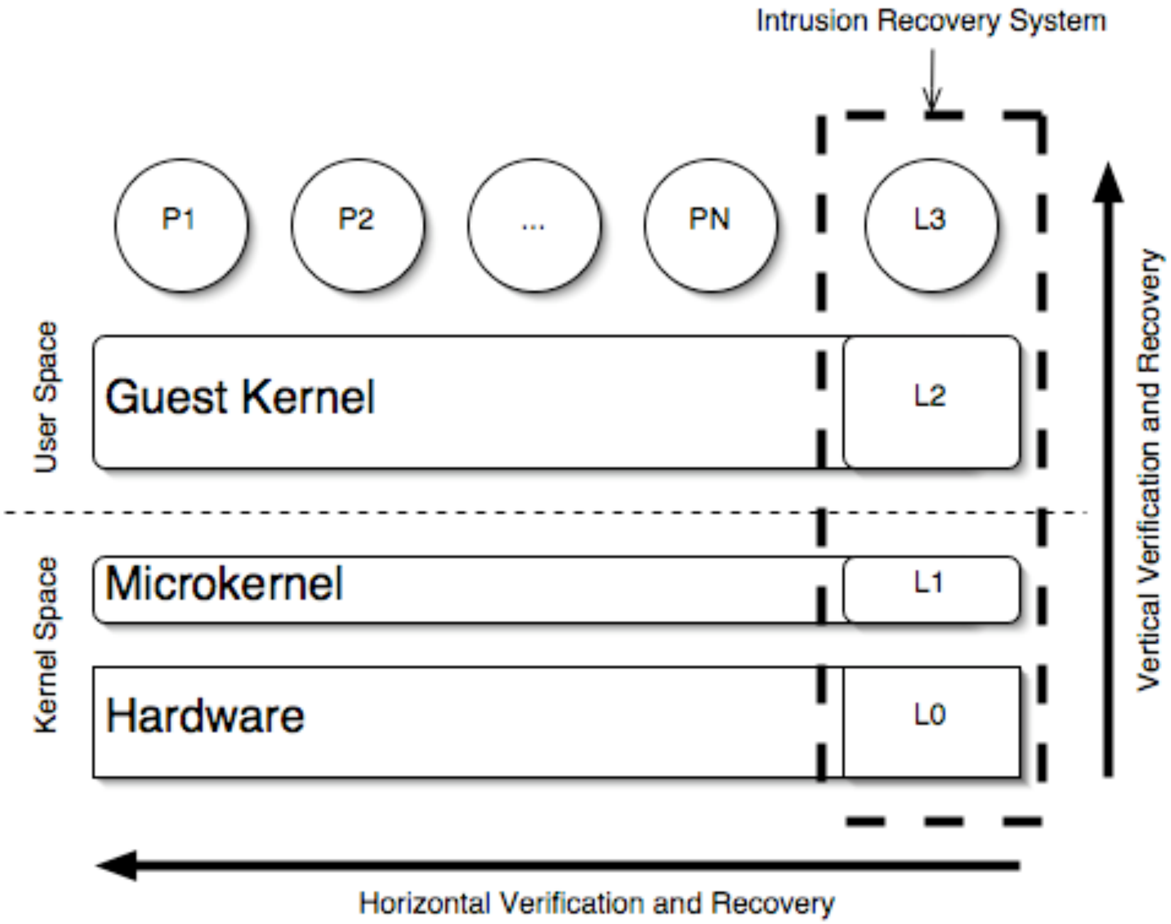
L4 Security Problems?

- Passing interrupt association
- Direct memory access
- Fill up page mapping database
- Kernel accessible on disk
- Cli/sti
- A few more…

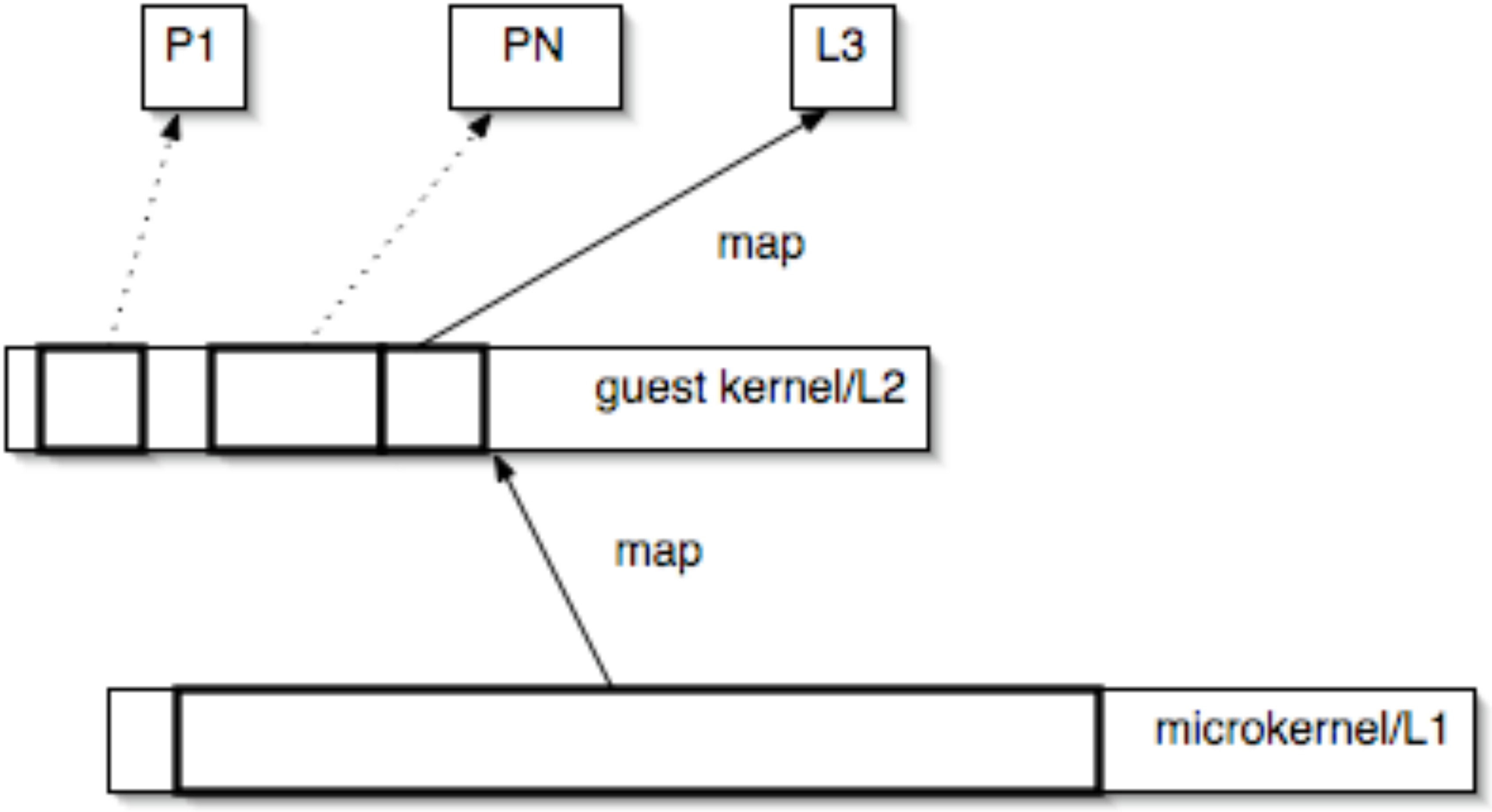
Talk Overview

- Rootkit background
- System call table tools
  - Demos
- L4 microkernel introduction
- **Spine architecture**
- Intrusion recovery system (IRS)
  - Demos
- Concluding remarks

Spine Architecture


Intrusion Recovery System
P1
P2
...
PN
L3
User Space
Guest Kernel
L2
Kernel Space
Microkernel
L1
Hardware
L0
Vertical Verification and Recovery
Horizontal Verification and Recovery

Memory Hierarchy Detail



Spine Architecture Details

- Uses L4 Fiasco microkernel
- L4Linux runs on top of microkernel
- User tasks run on L4Linux
- Intrusion recovery system consists of levels 0 through 3

L4Linux

- Port of Linux kernel to L4 architecture
- “paravirtualization” vs. pure virtualization
- Linux kernel runs in user space
- Binary compatible

Talk Overview

- Rootkit background
- System call table tools
  - Demos
- L4 microkernel introduction
- Spine architecture
- **Intrusion recovery system (IRS)**
  - **Demos**
- Concluding remarks

Intrusion Recovery System

- Capable of recovering from rootkit installations
- Maintain a copy of known good state to verify system integrity and repair if needed
- Must be integral part of operating system

IRS Cont…

- Intrusion detection system is part of IRS
  - Must be able to detect that an intrusion has occurred in order to recover from it
- Most difficult part of problem is verifying system integrity
  - How to verify data structures, config files, etc.
- Another important challenge is verifying integrity of IRS itself
  - Malware has been known to disable IDS’s

Multi-Level IRS Reasoning

- Difficult to monitor state of entire system from one vantage point
- Difficulty comes in bridging the semantic gap between layers of the system
- We use a multi-level approach

Multi-Leveled IRS Detail

- **L3** - verify file system state and repair if needed
- **L2** - kernel module to verify integrity of L4Linux and L3 and repair if needed
- **L1** - microkernel modifications to verify state of L2 and repair if needed; also provides secure storage for known good state
- **L0** - hardware support for maintaining isolation and verifying L1 (more hardware needed)

Demos

Intrusion Recovery System Demonstration

Talk Overview

- Rootkit background
- System call table tools
  - Demos
- L4 microkernel introduction
- Spine architecture
- Intrusion recovery system (IRS)
  - Demos
- **Concluding remarks**

Limitations and Conclusions

- Can an attacker install a microkernel-level rootkit?
- What if attacker has physical access?
- There is no be all end all solution! However, an IRS can make systems more reliable.

Thanks!

- Henry Owen
- John Levine
- Sven Krasser
- Greg Conti

Links

[Network and Security Architecture website]
http://www.ece.gatech.edu/research/labs/nsa/index.shtml

[Georgia Tech Information Security Center]
http://www.gtisc.gatech.edu/

[Fiasco project]
http://os.inf.tu-dresden.de/fiasco/

[Xen]
http://www.cl.cam.ac.uk/Research/SRG/netos/xen/

[Samhain Labs]
http://la-samhna.de

[Chkrootkit]
http://www.chkrootkit.org

[DaWheel, “So you don’t have to reinvent it!”]
http://www.dawheel.org

Questions?

Julian Grizzard

grizzard AT ece.gatech.edu