CS 194-1 (CS 161)
Computer Security

Lecture 23

Operating System Security; Rootkits

November 27, 2006
Prof. Anthony D. Joseph
http://cs161.org/

Goals for Today

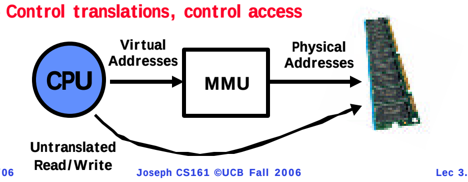
- Operating System security mechanisms
  - – Keep malicious programs from crashing OS
  - – Keep malicious programs from crashing each other
  - – Hardware helps isolate a program's effects to within just that program
    - » Address translation with non-executable regions
    - » Dual mode operation
- Rootkits
  - – Definition and history
  - – User-mode rootkits
  - – Kernel module/hooking rootkits
- Control over what applications run on a platform
  - – Need a secure environment from HW to OS levels

Operating System Security

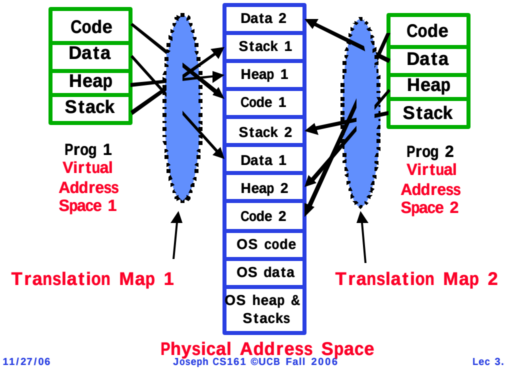
- Simple Policy:
  - – Don't allow programs to read/write memory of other programs or of the Operating System
- What is an Address Space?
  - – All the memory addresses a program can touch
    - » All the state that a program can affect or be affected by
  - – Each program (process) and kernel has potentially different address spaces.
- Achieve protection by restricting what a program can touch!
- Address Translation:
  - – Translate from Virtual Addresses (emitted by CPU) into Physical Addresses (of memory)

Address Translation

- Mapping *often* performed using table lookup in Hardware by Memory Management Unit (MMU)
  - – Separate table for each user address space
  - – No way for a program to even talk about other program's addresses
- Translation also helps with issue of stuffing multiple programs into memory
- Translation helps protection:
  - – Control translations, control access



Address Translation Example



Dual Mode Operation

- Should Users be able to change Page Table???
- Hardware provides at least two modes:
  - – "Kernel" mode (or "supervisor" or "protected")
  - – "User" mode: Normal programs executed
- Some instructions/ops prohibited in user mode:
  - – Example: cannot modify page tables in user mode
    - » Attempt to modify Þ Exception generated
- Transitions from user mode to kernel mode:
  - – System Calls, Interrupts, Other exceptions

Lock User-Programs in Asylum

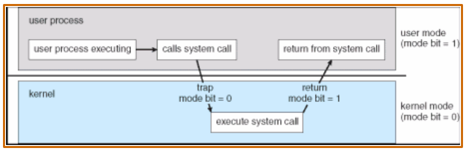
- Idea: Lock user programs in padded cell with no exit or sharp objects
  - Cannot change mode to kernel mode
  - User cannot modify page table mapping
  - Limited access to memory: cannot adversely effect other processes
    - » Side-effect: Limited access to memory-mapped I/O operations (I/O that occurs by reading/writing memory locations)
  - Limited access to interrupt controller
  - What else needs to be protected?
- A couple of issues
  - How to share CPU between kernel and user programs?
    - » Kinda like both the inmates and the warden in asylum are the same person. How do you manage this???
  - How do programs interact?
  - How does one switch between kernel and user modes?
    - » OS ® user (kernel ® user mode): getting into cell
    - » User® OS (user ® kernel mode): getting out of cell

How to get from Kernel® User

- What does the kernel do to create a new user process?
  - Allocate and initialize address-space control block
  - Read program off disk and store in memory
  - Allocate and initialize translation table
    - » Point at code in memory so program can execute
    - » Possibly point at statically initialized data
  - Run Program:
    - » Set machine registers
    - » Set hardware pointer to translation table
    - » Set processor status word for user mode
    - » Jump to start of program
- How does kernel switch between processes?
  - Same saving/restoring of registers as before
  - Save/restore PSL (hardware pointer to translation table)

User® Kernel (System Call)

- Can't let inmate (user) get out of padded cell on own
  - Would defeat purpose of protection!
  - So, how does the user program get back into kernel?



- System call: Voluntary procedure call into kernel
  - Hardware for controlled User®Kernel transition
  - Can any kernel routine be called?
    - » No! Only specific ones.
  - System call ID encoded into system call instruction
    - » Index forces well-defined interface with kernel

System Call Continued

- What are some system calls?
  - open, close, read, write, lseek, delete, mkdir, rmdir, truncate, chown, chgrp, fork, exit, wait (like join)
  - Network: socket create, set options
- Are system calls constant across operating systems?
  - Not entirely, but there are lots of commonalities
  - Also some standardization attempts (POSIX)
- What happens at beginning of system call?
    - » On entry to kernel, sets system to kernel mode
    - » Handler address fetched from table/Handler started
- System Call argument passing:
  - In registers (not very much can be passed)
  - Write into user memory, kernel copies into kernel mem
    - » User addresses must be translated!
    - » Kernel has different view of memory than user
  - Every argument must be explicitly checked!
    - » TOCTTOU vulnerabilities!

User® Kernel (Exceptions)

- A system call instruction causes a synchronous exception (or "trap")
  - In fact, often called a software "trap" instruction
- Other sources of synchronous exceptions:
  - Divide by zero, Illegal instruction, Bus error (bad address, e.g. unaligned access)
  - Segmentation Fault (address out of range)
  - Page Fault (for illusion of infinite-sized memory)
- Interrupts are Asynchronous Exceptions
  - Examples: timer, disk ready, network, etc….
  - Interrupts can be disabled, traps cannot!
- On system call, exception, or interrupt:
  - Hardware enters kernel mode with interrupts disabled
  - Saves PC, then jumps to appropriate handler in kernel
  - For some processors (x86), processor also saves registers, changes stack, etc.
- Actual handler typically saves registers, other CPU state, and switches to kernel stack

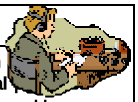
Context Switching

- Switching from one process to another one
  - Save hardware pointer to process A's translation table
  - Load hardware pointer to process B's translation table
- Now that we have isolated processes, how can they communicate?
  - Two models: shared memory, or via kernel

Administrivia

- Homework 3 due 12/1
- Midterm 2
  - Grades will be posted today or tomorrow
  - Will be handed back Thursday in section
- Midterm 3 is in-class on 12/6
  - In-class review 12/4

Communication



- Two models for interprocess communication
  - Shared memory: common mapping to physical
    - » As long as we place objects in shared memory address range, threads from each process can communicate
    - » Note that processes A and B can talk to shared memory through different addresses
    - » In some sense, this violates the whole notion of protection that we have been developing
  - If address spaces don't share memory, all inter-address space communication must go through kernel (via system calls)
    - » Byte stream producer/consumer (put/get): Example, communicate through pipes connecting `stdin/stdout`
    - » Message passing (send/receive): Will explain later how you can use this to build remote procedure call (RPC) abstraction so that you can have one program make procedure calls to another
    - » File System (read/write): File system is shared state!
- Be careful to avoid TOCTTOU vulnerabilities!

HW Support to Detect Buffer Overflow

- Add flag bits to each Page/Segment Table Entry
  - Mark individual memory areas as non-executable
    - » No Execute (NX) support (AMD Opteron and Athlon 64), Execute Disable (XD) support (Intel x86), Alpha, SPARC, PowerPC, Itanium, …
- Requires OS support to mark stack/heap as non-exec
  - Linux and Sun's Sparc/Solaris, Windows XP SP2
  - Any attempts to execute code from pages marked as non-executable results a program exception
- Does this prevent buffer overflow exploits?
  - No – only prevents buffer overflow exploits that try to execute code they send
  - Can overwrite return PC and execute an existing procedure (e.g., payload with return address for execve and some malicious parameters)

You've Been Owned!

- How can you tell when your machine has been compromised or taken over?
- "Odd" processes
- "Odd" windows
- "Extra" files
- Changed registry/configuration files
- "Extra" network connections, open ports
- …

What Is a Rootkit?

- Software or techniques that attempts to hide cracker's software from detection
  - Cracker's software can be anything
- Simple methods
  - Delete entries from login records, shell history
    - » Then, `last` command won't show intruder
- Cloaking methods (aka Ghostware)
  - Hide executables, libraries, config files, processes, …
    - » Hide from `ls`, `dir`, `ps`, `taskmgr`, …

Rootkit Functions

1. Maintain access
2. Attack local or other systems
3. Destroy evidence

- *Which OS'es are vulnerable?*

Maintaining Access

- Backdoor: telnet, rsh, ssh, irc, custom
  - UDP/TCP/ICMP protocol running on “high” port
  - Could require activation by “magic” TCP/IP packet, be a stealthy network sniffer, or use a covert channel, …
- Outbound connection
  - Works behind firewalls, no open inbound port to detect
  - Can be tunneled over outbound port 80

Attacking Local or Other Systems

- Collect local information
- Install network sniffer
- Perform DDoS attack
- Attempt to propagate
- …

Destroying Evidence

- Execute a log cleaner
- Hide its files
- Hide its processes
- Hide its network connections
- …

How Rootkits Get On Your Machine

- Cracker scans for vulnerable hosts
  - Or uses privilege elevation exploit
  - Or uses a worm or virus payload
- Exploits vulnerability to gain shell access
- Then copies over and installs rootkit …
  - Hides existence, records
  - Modifies start files
  - Starts daemon

Some Rootkit History Highlights

- 1989: First log cleaners found on hacked systems
- 1994: Early SunOS kits detected
- 1996: First Linux rootkits released
- 1997: Linux Kernel Module Trojans proposed
- 1998
  - Non-LKM kernel patching proposed
  - “Cult of the Dead Cow” created Windows rootkit “Back Orifice”
- 1999
  - Adore LKM kit released by TESO
  - “Cult of the Dead Cow” releases BO2K
- 2000: T0rn rootkit released
- 2002: Sniffer backdoors start to show up in kits

Pre-Rootkits: Hiding Login Events

- Many systems display a user’s last login time when they login
- Early crackers covered their tracks by using tools to modify login and other db records
  - Modify or delete `wtmp` file
  - Kill `syslogd`, and modify or delete `syslog.conf`
- How to defend systems?
  - Use a remote `syslogd`
  - But, some tools report remote entries in `syslog.conf`

Binary Library Rootkits: T0rn v8

- User-mode rootkit
- Easy to use (precompiled binaries)
  - Just type `./t0rn`.
  - Includes a log cleaner called `t0rnsb`
  - Also a network sniffer named `t0rns` and a log parser called `t0rnp`
- Replaces the tools that would show the rootkit:
  - `/usr/bin/du, /usr/bin/find, /sbin/ifconfig, /usr/sbin/in.fingerd, /bin/login, /bin/ls, /bin/netstat, /bin/ps, /usr/bin/sz, /usr/bin/top`
- Replaces system dynamic libraries to hide rootkit

Detecting T0rn v8

- Several serious implementation errors:
  - Different output from `ps` –eb than real one
  - Running `netstat` causes segmentation fault
- Wrong file sizes versus real files
- Easy to detect with `lsof` (list open files/ports)
  - Shows daemon listening on t0rn's default port
  - Shows all processes running under t0rn daemon (since it has open files)
- Can also be remotely detected
  - Use `nmap` to detect open ports
  - This is a common detection mechanism for non-stealthy rootkits
- Libraries only work for dynamically linked programs

BREAK

Kernel Module-based Rootkits

- Target Linux, Free/OpenBSD and Solaris
- Hook into the system kernel and replace/remap or modify/intercept) various system calls
  - Ones used by file system tools, and core kernel components
- Operating system core is no longer trustworthy
- Config file or built-in filename regexps lists files to hide:
  - Its own files, process, and sub-processes
  - Any of its inbound/outbound network connections (by address, protocol, listening process)

Detecting Kernel Module Rootkits

- Challenge is detection "from within the box"
  - Rootkit controls the vertical and the horizontal
- Leverage implementation errors
- Look for inconsistencies between different views
  - Can use cryptographic hashes of all important files (but have to protect hash values…)
  - Use tcsh's built-in ls: ls-F
  - Compare results from lower level interface
- Ideal solution:
  - Compare against known good system or CDROM
    - » Boot from CDROM/remote system and then examine disk

User-Mode Windows Rootkit: Back Orifice

- Windows is also vulnerable to user and kernel rootkits…
- Back Orifice (Win98 and WinNT systems)
  - Hid by running as a "system service"
  - Modified a registry startup entry
  - Listened for remote commands
  - Wasn't very stable under WinNT
- Didn't really try to hide itself
  - Was visible to process list tools

Kernel Module Windows Rootkit: BO2K

- Similar behavior as Unix kernel rootkits
  - Targeted W2K systems
- Installed itself into kernel memory
- Hooked kernel functions with its own modified functions
  - Blocked filesystem, process table and other attempts to find BO2K

Detecting Windows Kernel Rootkits

- Examine startup registry entries
  - Works for many rootkits
- In the box checks
  - Compare Win32 API results with results from low level kernel calls (e.g., process list, master file table,…)
  - Compare cryptographic hashes against known correct values
  - Look for hiding actions (create file/dir with prefixes)
- Out of the box checks
  - Compare against known good media/system

Rooting a Windows Kernel Rootkit

- Microsoft Research Tricks for using rootkit against itself
- Same name attack
  - Copy cmd.exe to same name/prefix as rootkit
  - Launch with start command
  - Rootkit can't hook itself, so built-in commands can run and see rootkit files, processes, directories, …
- Tools same name attack
  - Pick tool of choice for removing rootkit
  - Use same name attack, as rootkit won't block itself

Kernel Hooking Abuses

- Many anti-virus, firewall, anti-spyware and other tools use kernel hooking tricks
  - Can affect system stability when multiple programs are hooking kernel
  - MS Vista will block unsigned program hooking
- Sony XCP used kernel hooking to hide itself
- Problem is that crackers may be able to exploit cloaking to hide their tools!

Summary

- OS Security mechanisms - hardware helps isolation
  - Address translation
  - Dual mode operation
  - New HW options: non-executable regions
- Rootkits – all systems are vulnerable
  - On going arms race between crackers and detection tools…
  - Out of the box detection will always be possible
  - In the box detection will increase in difficulty