ELEC 377 – Operating Systems

Week 12 – Class 2

Admin

- Lab 4/5 Will be marked shortly
- Quiz #3 returning today

Today

- Unix History

What is a Root Kit?

- Root Kit is software to hide the evidence of system modification
- Originally used by intruders in Unix systems to hide changes to systems
  - ◊ add a back door process such as a chat daemon or ftp server running on non-standard port
  - ◊ changes to ps, netstat, w, passwd and other system commands to hide the back door
- Now applies to any operating system
  - ◊ Changes are now usually made to kernel and system libraries rather than to system commands
    - – Although some combine both system libraries and system commands

What is a Root Kit?

- Not the initial vulnerability
  - ◊ initial vulnerability is used to gain access, root kit is used to maintain access to compromised system
  - ◊ Sometimes the intruder patched vulnerability to keep ‘exclusive’ access to the system
  - ◊ root kit may attempt to maintain ownership of the system

    - one part of root kit notices when another part has been removed and reinstalls that component
- Often used by viruses and worms to disguise activities.
  - ◊ Thus rootkit detection is a concern for Security Vendors.

Legal Implications Canadian Laws

- Several attempts to introduce legislation (C-11)
  - ◊ Several common themes
- Outlaws circumvention of TPM or distribution of circumvention techniques (similar to DMCA)
  - ◊ Some exceptions :
- Other issues with legislation
  - ◊ no fair-dealing rights for anything protected by TPM

http://www.michaelgeist.ca

Other News

- Sony/BMG exec moves to MediaMax CEO
  - ◊ Kevin M. Clement former Senior Director, New Technology of SONY/BMG
- Gartner Group
  - ◊ Data partition can be disabled with a piece of tape. (DMCA violation?)

Rootkits in Linux

- How would one accomplish this in Linux?
- system calls use int 0x80
  - ◊ system call number in eax
  - ◊ sys_call_table points to system call handler
  - ◊ modules can modify sys_call_table entries to point to them
- create, new, read directory, open file routines
- lsmod uses /dev/kmem to scan a list
  - – remove module from list
- Modify /proc drivers not to show the processes belonging to the back door the root kit is hiding
- put processes in /etc/rc/init.d to ensure they start up each time - (ls hides the files...)

Root Kit Research

- Commercial and Personal Systems
  - ◊ when you get malware, you want to remove it
  - ◊ limit its damage
- Sensitive Systems.
  - ◊ You don’t want to eradicate the malware
  - ◊ You need to observe it
    - -- who is it reporting to?
    - -- what kind of information is it interested in
    - -- limit access to sensitive information
      - ◊ Problem: it is checking to see if anyone is watching
    - -- may self destruct/or may attempt to destroy system.
    - -- may change its behaviour.

Sensitive Systems

- Counter-Intelligence Operations
  - ◊ after detecting malware, you provide a simulated environment (including new operator) -- research on fake operator!!
  - ◊ replace systems it has access to, with fake systems with fake information
- Observe the malware
  - ◊ CASCON paper
  - ◊ Use root kit techniques to hide the anti malware software from the malware
  - ◊ Installed at time OS is installed -- we are in first!!

Sensitive Systems

- Battle of limited resources
  - ◊ the malware is trying to remain covert
  - ◊ covert channels to get data out to handler
  - ◊ limited access to CPU time and Memory

  -- consume to many resources, then becomes obvious you are there...
- We are also trying to remain covert
  - ◊ However, we are there first
  - ◊ they have to use limited resources to both look for us and to carry out primary mission (obtain and exfiltrate desired information)

Root Kit Research - Our Rootkit

- Kernel Level Asynchronous Procedure Calls(APC)
  - ◊ threads and processes can register a call back routine

    -- attached to an event such as a key press, or a timer
  - ◊ Available to kernel threads
  - ◊ Higher priority threads can attach callbacks to lower priority threads

Root Kit Research

- Kernel Level Asynchronous Procedure Calls(APC)
  - ◊ We start with high priority
    - -- during init, allocate a memory block and copy ourselves into it, register a callback on another thread.
  - ◊ call back executes with knowledge of the thread virtual memory tables, and other process info
  - ◊ Our anti-malware executes entirely as APC callbacks.
  - ◊ copy to different memory location
  - ◊ register callbacks on different threads
  - ◊ Can inject into malware’s thread and look at malware in malware’s context
  - ◊ jump onto firefox thread to exfiltrate information

Intrusion Detection

- Aspects
  - ◊ real time vs after intrusion
  - ◊ what is examined (commands, system calls, network packets, etc.)
  - ◊ response

- What is an Intrusion?
  - ◊ signature based detection

    - virus, multiple login attempts
  - ◊ anomaly based detection

    - something not normal

Intrusion Detection

- Issues
  - ◊ Delay in adding signatures
  - ◊ Errors in signatures
    - AVG accidentally removes user32.dll

  - ◊ stealth channels
    - some intruders only want limited information
    - other want to stay and spy a while....

Intrusion Detection

- Audits and Logs
  - ◊ UNIX syslog daemon
  - ◊ most daemons use the syslog daemon to log activities
  - ◊ swatch - scans daemons for anomalous activity

- Tripwire
  - ◊ Purdue University
  - ◊ checksum of system files and attributes
    - detect modifications
  - ◊ detect modification of tripwire?

Security is Increasingly Important

- Continue to be interesting in ways never thought of before

  - photo of keys??

    - can now cut keys from keys appearing in a picture, even from a distance of 200 feet

Unix - History

- 1969
  - PDP-7 (Assembly Language)
  - File-centric view of the world
  - Small group operating system
  - C - developed to write UNIX on PDP-11
  - Given away to Universities with Source (1976)
  - Language Design and Programming Methodology Conference (1979)
  - Lyon’s book
  - Ported to many different architectures
- 1991
  - Linux
  - Free version of Unix for x86

Unix - Kernel

- Minimal Kernel
  - ◊ small address space (< 64K bytes)
  - ◊ some things were implemented as user processes (glob)
  - ◊ better hardware -> larger kernel
  - ◊ small tool centric view of the world
  - ◊ Early kernels (both Unix and Linx) were monolithic (one large program). Installation involved building the kernel for the given hardware
  - ◊ Extended with loadable modules/device drivers

Unix - Scheduling

- Priority based scheduling
  - ◊ all processes at a given priority level are scheduled round robin.
  - ◊ Processes priorities are aged by the kernel
  - ◊ Extended with soft real time scheduling
  - ◊ no longer simple.

Linux Processes Scheduling

- two algorithms
  - ◊ priority based scheduling
  - ◊ real-time scheduling
  - ◊ part of process personality

- priority based schedule
  - ◊ credit based algorithm
  - ◊ each timer interrupt (jiffy), the current process looses one credit
  - ◊ process in ready queue with most credits goes next
  - ◊ what happens when all process in ready queue are out of credits?

Linux Processes Scheduling

- credit rebalancing
  - ◊ all process in the ready queue are out of credits
  - ◊ processes in wait queues may still have credits
  - ◊ generate new credits for every process (not just ready queue processes)

    credits = credits / 2 + priority

◊ mixes priority of process and process history

- processes with a lot of wait time accumulate credits and always run when ready
- CPU bound processes always short on credits

O(1) Scheduler

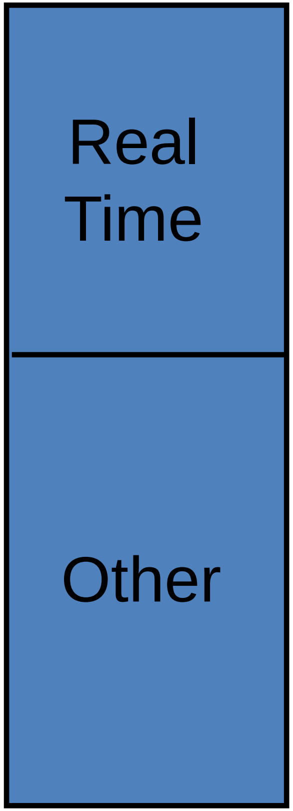
- Kernel 2.5...

◊ recalculating credits means a computation for every process in the system.
- fine for small systems with a small number of processes
- overhead of a context switch grows as the number of ready processes grow
- bottleneck for SMP, Java (native thread model)

◊ New algorithm created
- constant time no matter how many processes are ready to run.
- better support for SMP (Symmetric multiprocessing)

O(1) Scheduler

0 High
.
.
.
99
100
.
.
.
140 Low



Real Time

Other

200 ms

10 ms

O(1) Scheduler

- tasks at a given priority are added at the end of each queue (a couple of pointer changes)
- Two sets of queues, Active and Expired
- Each process is initially given a time allocation based on priority.
- As it executes, time is subtracted from the allocation
- When empty, time slice is recalculated and the process is put on the expired set of queues.
- If the active queue for a given priority is empty, then the it is swapped with the expired queue.

Dynamic Priorities

- Lower priority of CPU bound
- Raise Priority of Upper Bound
- Interactivity heuristic compares sleep time to run time
- +/5 priority points (changes which queue)

UNIX - file centric view

- Much of the Unix Kernel Design is visible in the file system
  - ◊ Hard disks merged into a single tree
    - some attempt to hide disks
    - not entirely successful
  - ◊ Start with single file
    - permissions
    - rwx - user, group and other
    - user ID, group ID (16 bits)
    - a user may belong to more than one group

UNIX - single file view

- Total of 16 attribute bits for any file
  - ◊ 9 so far (rwx)
  - ◊ s - set uid bits (x 2)
  - ◊ d - directory bit
  - ◊ l - symbolic link
  - ◊ b - block special device
  - ◊ c - character special device
  - ◊ p - pipe bit
  - some combinations illegal
    - ◊ d and b for example

UNIX - single file view

- User and Group

◊ processes are executed by owner and a group
- just like files, have an owner and a group
- users can change their current group
- used for accessing resources
- most resources accessed through the file system
- file permission bits determine resource access

◊ setuid bits permit owner of command to determine the user id and group id of process
- *effective* uid (euid), *effective* group id (egid)
- most versions of unix permit the executable to switch between real and effective user and group ids (some allow for root only)

UNIX - single file view

- setuid bits
  - ◊ possible security hole

    - paths with ‘.’ at start
  - ◊ real uses

    - passwd command

    - database access

    - data file access

UNIX - single file view

- directory bit
  - ◊ file is a directory
  - ◊ can be read as a regular file
  - ◊ opendir, closedir, readdir, link, unlink
  - ◊ other bits have different meanings
    - - r bit - read directory contents
    - - x bit traverse directory
      - - system must read
      - - r no x means can open contents if name known
    - - s bit
      - - cannot execute directory
      - - set gid bit is used to preserve group of directory

UNIX - single file view

- link bit
  - ◊ file contains a single line which is a path (relative or absolute) to another directory or file
    - permits links to cross file systems
    - share directories
- b and c bits
  - ◊ device files (/dev)
  - ◊ files do not contain any data blocks
  - ◊ used for processes talk directly to device drivers
  - ◊ major and minor modes
    - major mode identifies the device driver
    - minor mode is parameter
    - linux hda = b 3 0, tty0 = c 4 0

Linux Processes

- first process started is *init*
  - ◊ reads /etc/inittab
  - ◊ starts any daemons
  - ◊ some programs are monitored for restart
- login process (character based)
  1. *init* starts *getty* on the terminal (console)
  2. *getty* puts up login prompt and waits for username
  3. *getty* spawns *login* with username as argument
  4. *login* asks for password (turns off echo)
  5. *login* spawns the shell given in /etc/passwd
  6. when shell exits, *init* starts another instance of *getty*