

# CS419 – Spring 2010

## Computer Security

Virtual Machines and Malware

Vinod Ganapathy  
Lecture 20

Readings: VMWare paper + Thompson's paper

# Course Review Forms

- Need one volunteer.
- Please turn in completed forms to the CS undergraduate office
  - Joanne Walsh or Komal Agarwal

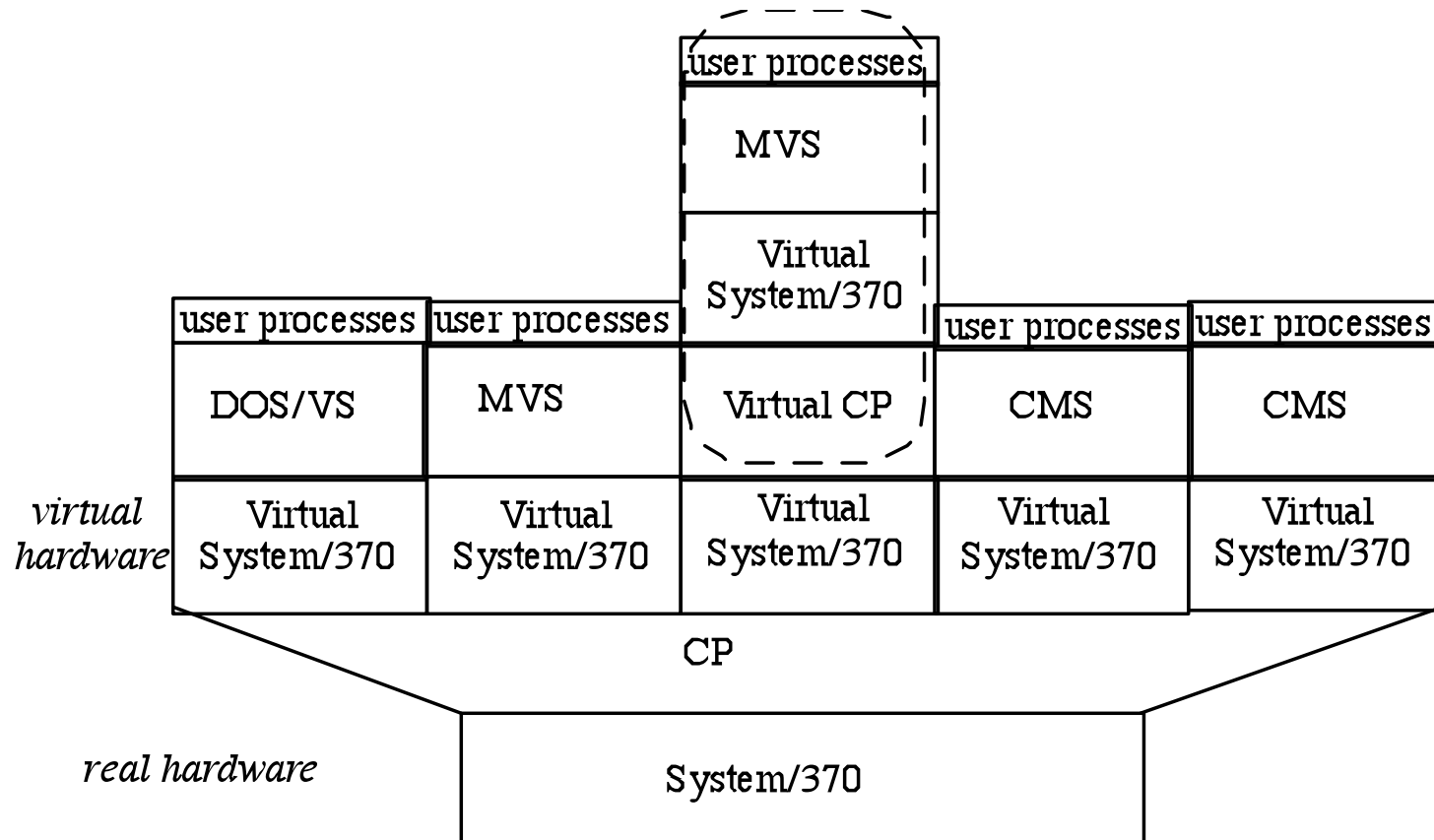
# Part 1: Virtual Machines

- Virtual Machine Structure
- Virtual Machine Monitor
  - Privilege
  - Physical Resources
  - Paging

# What Is It?

- *Virtual machine monitor* (VMM) virtualizes system resources
  - Runs directly on hardware
  - Provides interface to give each program running on it the illusion that it is the only process on the system and is running directly on hardware
  - Provides illusion of contiguous memory beginning at address 0, a CPU, and secondary storage to *each* program

# Example: IBM VM/370



Adapted from Dietel, pp. 606-607

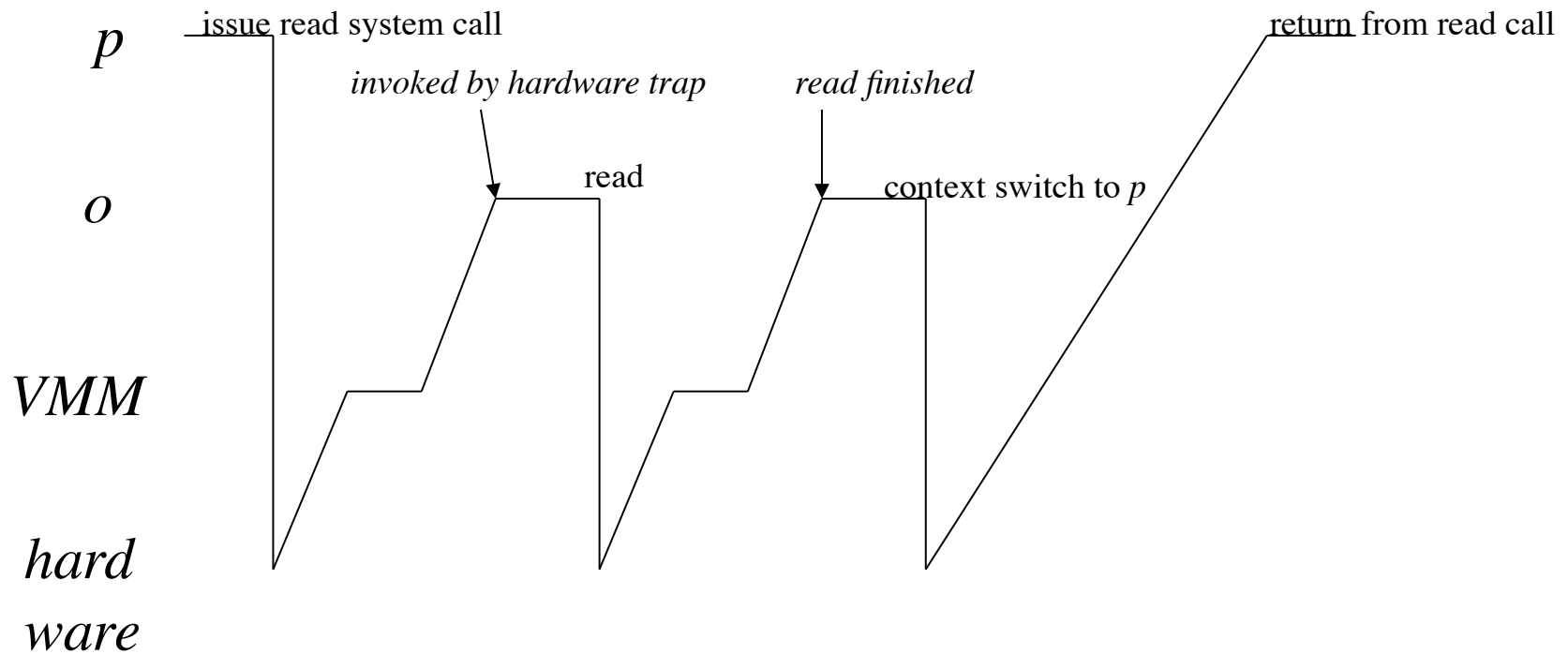
# Privileged Instructions

1. VMM running operating system  $o$ , which is running process  $p$ 
  - $p$  tries to read—privileged operation traps to hardware
2. VMM invoked, determines trap occurred in  $o$ 
  - VMM updates state of  $o$  to make it look like hardware invoked  $o$  directly, so  $o$  tries to read, causing trap
3. VMM does read
  - Updates  $o$  to make it seem like  $o$  did read
  - Transfers control to  $o$

# Privileged Instructions

4.  $o$  tries to switch context to  $p$ , causing trap
5. VMM updates virtual machine of  $o$  to make it appear  $o$  did context switch successfully
  - Transfers control to  $o$ , which (as  $o$  apparently did a context switch to  $p$ ) has the effect of returning control to  $p$

# Privileged Instructions



# Privilege and VMs

- *Sensitive instruction* discloses or alters state of processor privilege
- *Sensitive data structure* contains information about state of processor privilege

# When Is VM Possible?

- Can virtualize an architecture when:
  1. All sensitive instructions cause traps when executed by processes at lower levels of privilege
  2. All references to sensitive data structures cause traps when executed by processes at lower levels of privilege

# Example: VAX System

- 4 levels of privilege (user, supervisor, executive, kernel)
  - CHMK changes privilege to kernel level; sensitive instruction
    - Causes trap *except* when executed in kernel mode; meets rule 1
  - Page tables have copy of PSL, containing privilege level; sensitive data structure
    - If user level processes prevented from altering page tables, trying to do so will cause a trap; this meets rule 2

# Multiple Levels of Privilege

- Hardware supports  $n$  levels of privilege
  - VM must also support  $n$  levels
  - VM monitor runs at highest level, so  $n-1$  levels of privilege left!
- Solution: virtualize levels of privilege
  - Called *ring compression*

# Example: VAX VMM System

- VMM at kernel level
- VMM maps virtual kernel and executive level to (real) executive mode
  - Called *VM kernel level*, *VM executive level*
  - Virtual machine bit added to PSL
    - If set, current process running on VM
  - Special register, VMPSL, records PSL of currently running VM
  - All sensitive instructions that *could* reveal level of privilege get this information from VMPSL or trap to the VMM, which then emulates the instruction

# Alternate Approach

- Divide users into different classes
- Control access to system by limiting access of each class

# Example: IBM VM/370

- Each control program command associated with user privilege classes
  - “G” (general user) class can start a VM
  - “A” (primary system operator) class can control accounting, VM availability, other system resources
  - “Any” class can gain or surrender access to VM

# Physical Resources and VMs

- Distributes resources among VMs as appropriate
  - Each VM appears to have reduced amount of resources from real system
  - Example: VMM to create 10 VMs means real disk split into 10 minidisks
    - Minidisks may have different sizes
    - VMM does mapping between minidisk addresses, real disk addresses

# Example: Disk I/O

- VM's OS tries to write to disk
  - I/O instruction privileged, traps to VMM
- VMM checks request, services it
  - Translates addresses involved
  - Verifies I/O references disk space allocated to that VM
  - Services request
- VMM returns control to VM when appropriate
  - If I/O synchronous, when service complete
  - If I/O asynchronous, when service begun

# Paging and VMs

- Like ordinary disk I/O, but 2 problems
  - Some pages may be available only at highest level of privilege
    - VM must remap level of privilege of these pages
  - Performance issues
    - VMM paging its own pages is transparent to VMs
    - VM paging is handled by VMM; if VM's OS does lots of paging, this may introduce significant delays

# Example: VAX/VMS

- On VAX/VMS, only kernel level processes can read some pages
  - What happens if process at VM kernel level needs to read such a page?
    - Fails, as VM kernel level is at real executive level
  - VMM reduces level of page to executive, then it works
    - Note: security risk!
      - In practice, OK, as VMS allows executive level processes to change to kernel level

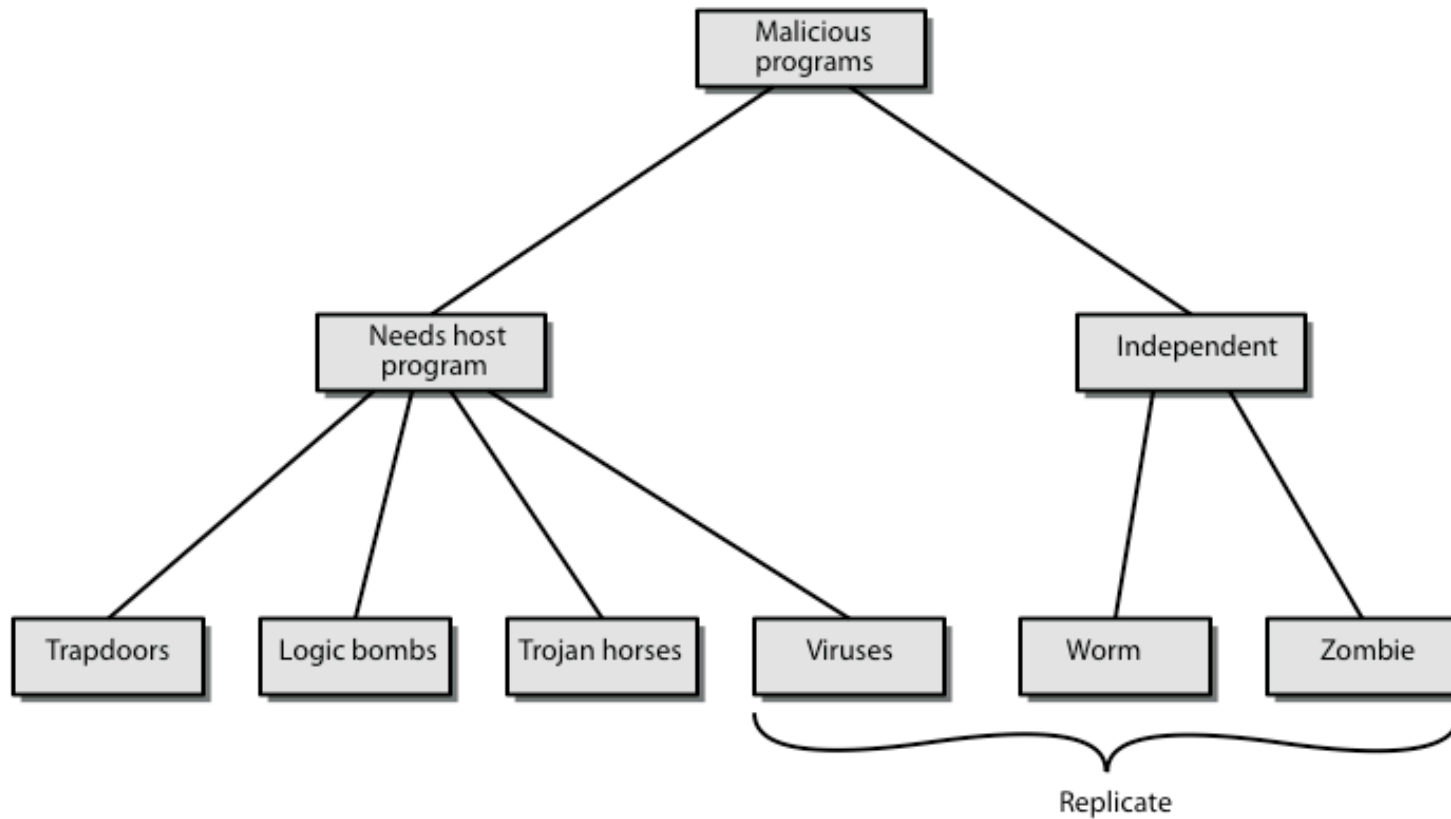
# Example: IBM VM/370

- Supports several different operating systems
  - OS/MFT, OS/MVT designed to access disk storage
    - If jobs being run under those systems depend on timings, delay caused by VM may affect success of job
  - If system supports virtual paging (like MVS), either MVS or VMM may cause paging
    - The VMM paging may introduce overhead (delays) that cause programs to fail that would not were the programs run directly on the hardware

# Part 2: Malware

- What is a:
  - Virus?
  - Worm?
    - Polymorphic viruses and worms?
  - Trojan horse?
  - Rootkit?
  - Bot?
  - Spyware program?

# Malicious Software



# Malicious Logic

- Shell script on a UNIX system:

```
cp /bin/sh /tmp/.xyzzzy  
chmod u+s,o+x /tmp/.xyzzzy  
rm ./ls  
ls $*
```

- Place in program called “ls” and trick someone into executing it
- You now have a setuid-to-*them* shell!

# Trojan Horse

- Program with an *overt* purpose (known to user) and a *covert* purpose (unknown to user)
  - Often called a Trojan
  - Named by Dan Edwards in Anderson Report
- Example: previous script is Trojan horse
  - Overt purpose: list files in directory
  - Covert purpose: create setuid shell

# Example: NetBus

- Designed for Windows NT system
- Victim uploads and installs this
  - Usually disguised as a game program, or in one
- Acts as a server, accepting and executing commands for remote administrator
  - This includes intercepting keystrokes and mouse motions and sending them to attacker
  - Also allows attacker to upload, download files

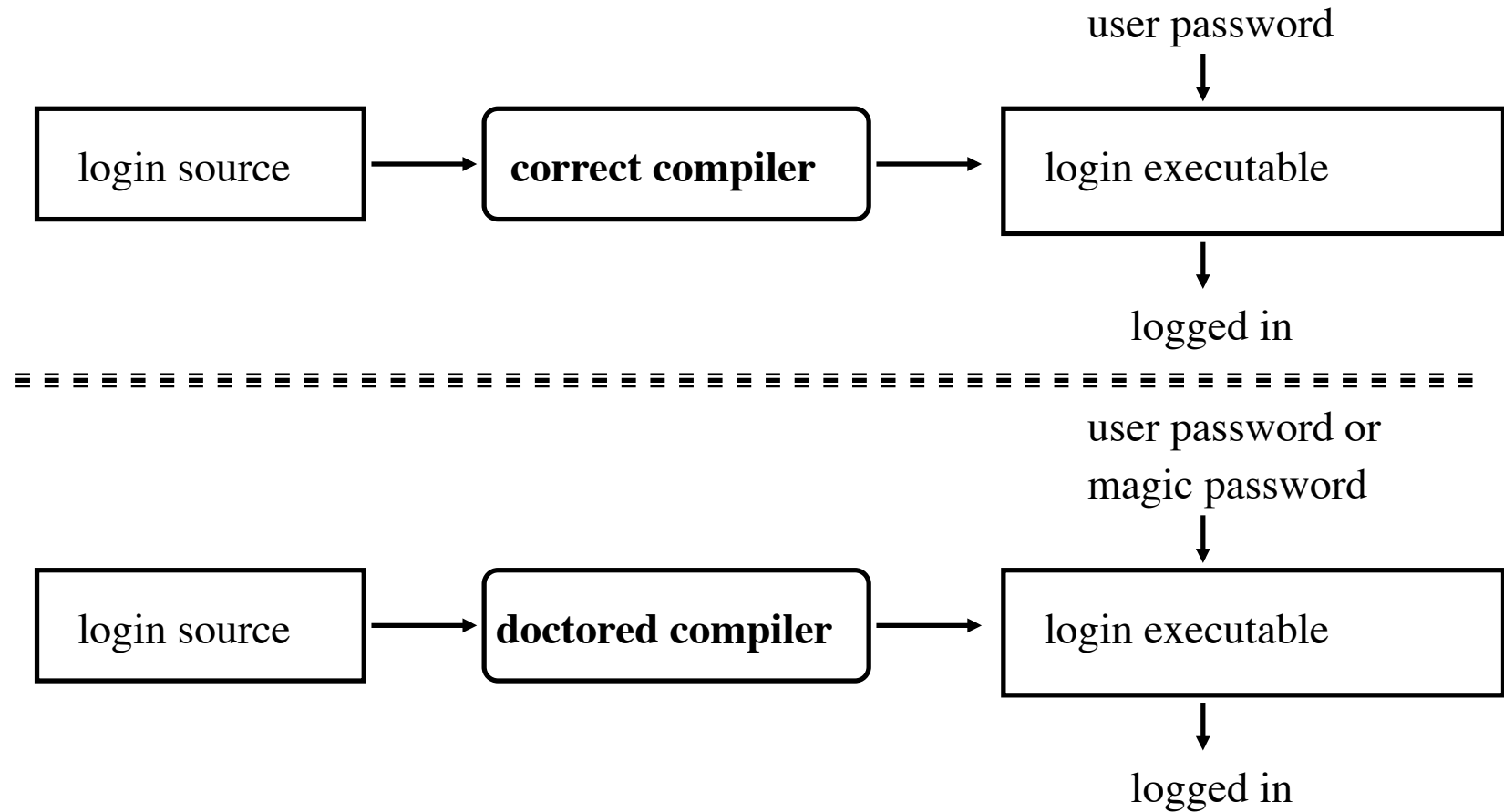
# Replicating Trojan Horse

- Trojan horse that makes copies of itself
  - Also called *propagating Trojan horse*
  - Early version of *animal* game used this to delete copies of itself
- Hard to detect
  - 1976: Karger and Schell suggested modifying compiler to include Trojan horse that copied itself into specific programs including later version of the compiler
  - 1980s: Thompson implements this

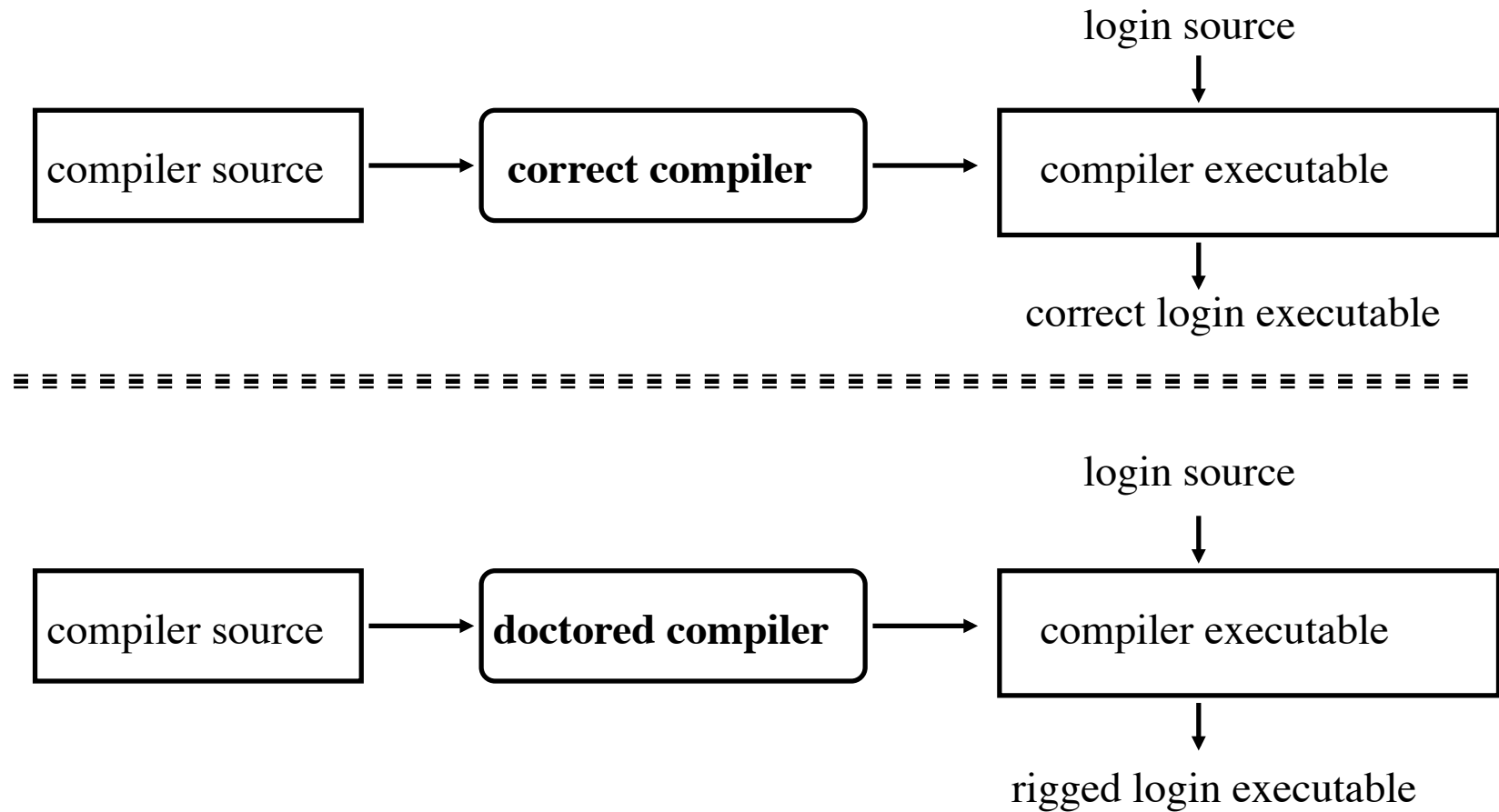
# Thompson's Compiler

- Modify the compiler so that when it compiles *login* , *login* accepts the user's correct password or a fixed password (the same one for all users)
- Then modify the compiler again, so when it compiles a new version of the compiler, the extra code to do the first step is automatically inserted
- Recompile the compiler
- Delete the source containing the modification and put the undoctored source back

# The Login Program



# The Compiler



# Comments

- Great pains taken to ensure second version of compiler never released
  - Finally deleted when a new compiler executable from a different system overwrote the doctored compiler
- The point: *no amount of source-level verification or scrutiny will protect you from using untrusted code*
  - Also: having source code helps, but does not ensure you're safe

# Backdoor or Trapdoor

- secret entry point into a program
- allows those who know access bypassing usual security procedures
- have been commonly used by developers
- a threat when left in production programs allowing exploited by attackers
- very hard to block in O/S
- requires good s/w development & update

# Logic Bomb

- one of oldest types of malicious software
- code embedded in legitimate program
- activated when specified conditions met
  - eg presence/absence of some file
  - particular date/time
  - particular user
- when triggered typically damage system
  - modify/delete files/disks, halt machine, etc

# Trojan Horse

- program with hidden side-effects
- which is usually superficially attractive
  - eg game, s/w upgrade etc
- when run performs some additional tasks
  - allows attacker to indirectly gain access they do not have directly
- often used to propagate a virus/worm or install a backdoor
- or simply to destroy data

# Zombie

- program which secretly takes over another networked computer
- then uses it to indirectly launch attacks
- often used to launch distributed denial of service (DDoS) attacks
- exploits known flaws in network systems

# Viruses

- a piece of self-replicating code attached to some other code
  - cf biological virus
- both propagates itself & carries a payload
  - carries code to make copies of itself
  - as well as code to perform some covert task

# Computer Virus

- Program that inserts itself into one or more files and performs some action
  - *Insertion phase* is inserting itself into file
  - *Execution phase* is performing some (possibly null) action
- Insertion phase *must* be present
  - Need not always be executed
  - Lehigh virus inserted itself into boot file only if boot file not infected

# Pseudocode

```
beginvirus:  
  if spread-condition then begin  
    for some set of target files do begin  
      if target is not infected then begin  
        determine where to place virus instructions  
        copy instructions from beginvirus to endvirus  
        into target  
        alter target to execute added instructions  
      end;  
    end;  
  end;  
  perform some action(s)  
  goto beginning of infected program
```

# History

- Programmers for Apple II wrote some
  - Not called viruses; very experimental
- Fred Cohen
  - Graduate student who described them
  - Teacher (Adleman) named it “computer virus”
  - Tested idea on UNIX systems and UNIVAC 1108 system

# First Reports

- Brain virus (1986)
  - Written for IBM PCs
  - Alters boot sectors of floppies, spreads to other floppies
- MacMag Peace virus (1987)
  - Written for Macintosh
  - Prints “universal message of peace” on March 2, 1988 and deletes itself

# More Reports

- Duff's experiments (1987)
  - Small virus placed on UNIX system, spread to 46 systems in 8 days
  - Wrote a Bourne shell script virus
- Highland's Lotus 1-2-3 virus (1989)
  - Stored as a set of commands in a spreadsheet and loaded when spreadsheet opened
  - Changed a value in a specific row, column and spread to other files

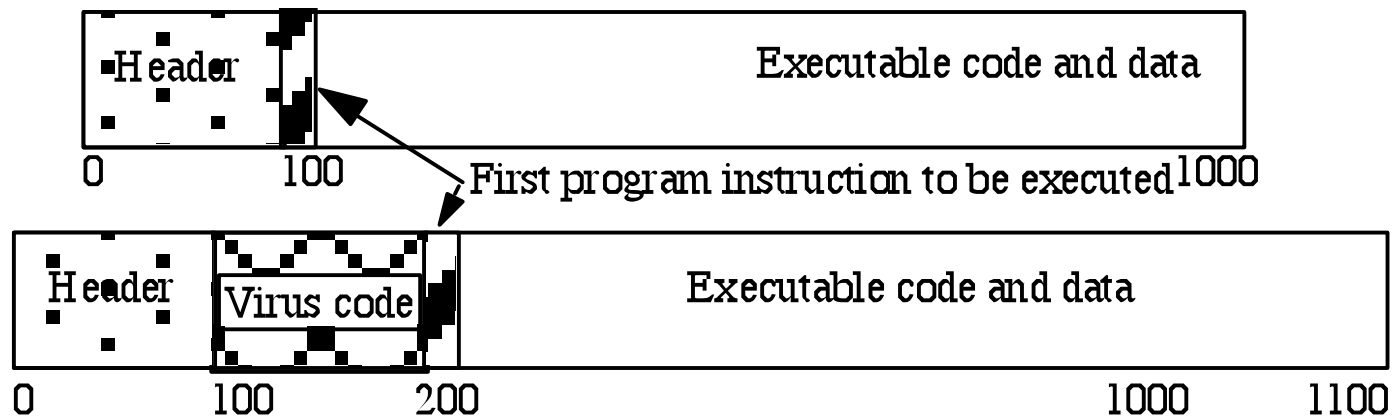
# Types of Viruses

- Boot sector infectors
- Executable infectors
- Multipartite viruses
- TSR viruses
- Stealth viruses
- Encrypted viruses
- Polymorphic viruses
- Macro viruses

# Boot Sector Infectors

- A virus that inserts itself into the boot sector of a disk
  - Section of disk containing code
  - Executed when system first “sees” the disk
    - Including at boot time ...
- **Example: Brain virus**
  - Moves disk interrupt vector from 13H to 6DH
  - Sets new interrupt vector to invoke Brain virus
  - When new floppy seen, check for 1234H at location 4
    - If not there, copies itself onto disk after saving original boot block

# Executable Infectors



- A virus that infects executable programs
  - Can infect either .EXE or .COM on PCs
  - May prepend itself (as shown) or put itself anywhere, fixing up binary so it is executed at some point

# Executable Infectors (*con't*)

- Jerusalem virus
  - Checks if system infected
    - If not, set up to respond to requests to execute files
  - Checks date
    - If not 1987 or Friday 13th, set up to respond to clock interrupts and then run program
    - Otherwise, set destructive flag; will delete, not infect, files
  - Then: check all calls asking files to be executed
    - Do nothing for COMND.COM
    - Otherwise, infect or delete
  - Error: doesn't set signature when .EXE executes
    - So .EXE files continually reinfected

# Multipartite Viruses

- A virus that can infect either boot sectors or executables
- Typically, two parts
  - One part boot sector infector
  - Other part executable infector

# TSR Viruses

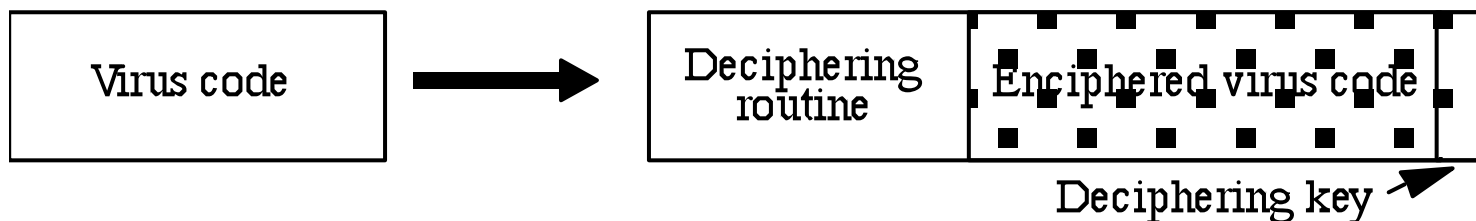
- A virus that stays active in memory after the application (or bootstrapping, or disk mounting) is completed
  - TSR is “Terminate and Stay Resident”
- Examples: Brain, Jerusalem viruses
  - Stay in memory after program or disk mount is completed

# Stealth Viruses

- A virus that conceals infection of files
- Example: IDF virus modifies DOS service interrupt handler as follows:
  - Request for file length: return length of *uninfected* file
  - Request to open file: temporarily disinfect file, and reinfect on closing
  - Request to load file for execution: load infected file

# Encrypted Viruses

- A virus that is enciphered except for a small deciphering routine
  - Detecting virus by signature now much harder as most of virus is enciphered



# Example

```
(* Decryption code of the 1260 virus *)
(* initialize the registers with the keys *)
rA = k1; rB = k2;
(* initialize rC with the virus;
   starts at sov, ends at eov *)
rC = sov;
(* the encipherment loop *)
while (rC != eov) do begin
    (* encipher the byte of the message *)
    (*rC) = (*rC) xor rA xor rB;
    (* advance all the counters *)
    rC = rC + 1;
    rA = rA + 1;
end
```

# Polymorphic Viruses

- A virus that changes its form each time it inserts itself into another program
- Idea is to prevent signature detection by changing the “signature” or instructions used for deciphering routine
- At instruction level: substitute instructions
- At algorithm level: different algorithms to achieve the same purpose
- Toolkits to make these exist (Mutation Engine, Trident Polymorphic Engine)

# Example

- These are different instructions (with different bit patterns) but have the same effect:
  - add 0 to register
  - subtract 0 from register
  - xor 0 with register
  - no-op
- Polymorphic virus would pick randomly from among these instructions

# Macro Viruses

- A virus composed of a sequence of instructions that are interpreted rather than executed directly
- Can infect either executables (Duff's shell virus) or data files (Highland's Lotus 1-2-3 spreadsheet virus)
- Independent of machine architecture
  - But their effects may be machine dependent

# Example

- Melissa
  - Infected Microsoft Word 97 and Word 98 documents
    - Windows and Macintosh systems
  - Invoked when program opens infected file
  - Installs itself as “open” macro and copies itself into Normal template
    - This way, infects any files that are opened in future
  - Invokes mail program, sends itself to everyone in user’s address book

# Computer Worms

- A program that copies itself from one computer to another
- Origins: distributed computations
  - Schoch and Hupp: animations, broadcast messages
  - Segment: part of program copied onto workstation
  - Segment processes data, communicates with worm's controller
  - Any activity on workstation caused segment to shut down

# Example: Christmas Worm

- Distributed in 1987, designed for IBM networks
- Electronic letter instructing recipient to save it and run it as a program
  - Drew Christmas tree, printed “Merry Christmas!”
  - Also checked address book, list of previously received email and sent copies to each address
- Shut down several IBM networks
- Really, a macro worm
  - Written in a command language that was interpreted

# Example: Internet Worm of 1988

- Targeted Berkeley, Sun UNIX systems
  - Used virus-like attack to inject instructions into running program and run them
  - To recover, had to disconnect system from Internet and reboot
  - To prevent re-infection, several critical programs had to be patched, recompiled, and reinstalled
- Analysts had to disassemble it to uncover function
- Disabled several thousand systems in 6 or so hours

# Rabbits, Bacteria

- A program that absorbs all of some class of resources
- Example: for UNIX system, shell commands:

```
while true
do
    mkdir x
    chdir x
done
```
- Exhausts either disk space or file allocation table (inode) space

# Logic Bombs

- A program that performs an action that violates the site security policy when some external event occurs
- Example: program that deletes company's payroll records when one particular record is deleted
  - The “particular record” is usually that of the person writing the logic bomb
  - Idea is if (when) he or she is fired, and the payroll record deleted, the company loses *all* those records

# Defenses

- Distinguish between data, instructions
- Limit objects accessible to processes
- Inhibit sharing
- Detect altering of files
- Detect actions beyond specifications
- Analyze statistical characteristics

# Data vs. Instructions

- Malicious logic is both
  - Virus: written to program (data); then executes (instructions)
- Approach: treat “data” and “instructions” as separate types, and require certifying authority to approve conversion
  - Keys are assumption that certifying authority will *not* make mistakes and assumption that tools, supporting infrastructure used in certifying process are not corrupt

# Information Flow Metrics

- Idea: limit distance a virus can spread
- Flow distance metric  $fd(x)$ :
  - Initially, all info  $x$  has  $fd(x) = 0$
  - Whenever info  $y$  is shared,  $fd(y)$  increases by 1
  - Whenever  $y_1, \dots, y_n$  used as input to compute  $z$ ,  $fd(z) = \max(fd(y_1), \dots, fd(y_n))$
- Information  $x$  accessible if and only if for some parameter  $V$ ,  $fd(x) < V$

# Example

- Anne:  $V_A = 3$ ; Bill, Cathy:  $V_B = V_C = 2$
- Anne creates program P containing virus
- Bill executes P
  - P tries to write to Bill's program Q
    - Works, as  $fd(P) = 0$ , so  $fd(Q) = 1 < V_B$
- Cathy executes Q
  - Q tries to write to Cathy's program R
    - Fails, as  $fd(Q) = 1$ , so  $fd(R)$  would be 2
- Problem: if Cathy executes P, R can be infected
  - So, does not stop spread; slows it down greatly, though

# Reducing Protection Domain

- Application of principle of least privilege
- Basic idea: remove rights from process so it can only perform its function
  - Warning: if that function requires it to write, it can write anything
  - But you can make sure it writes only to those objects you expect

# Trusted Programs

- No VALs applied here
  - UNIX command interpreters
    - csh, sh
  - Program that spawn them
    - getty, login
  - Programs that access file system recursively
    - ar, chgrp, chown, diff, du, dump, find, ls, restore, tar
  - Programs that often access files not in argument list
    - binmail, cpp, dbx, mail, make, script, vi
  - Various network daemons
    - fingerd, ftpd, sendmail, talkd, telnetd, tftpd

# Guardians, Watchdogs

- System intercepts request to open file
- Program invoked to determine if access is to be allowed
  - These are *guardians* or *watchdogs*
- Effectively redefines system (or library) calls

# Sandboxing

- Sandboxes, virtual machines also restrict rights
  - Modify program by inserting instructions to cause traps when violation of policy
  - Replace dynamic load libraries with instrumented routines

# Example: Race Conditions

- Occur when successive system calls operate on object
  - Both calls identify object by name
  - Rebind name to different object between calls
- Sandbox: instrument calls:
  - Unique identifier (inode) saved on first call
  - On second call, inode of named file compared to that of first call
    - If they differ, potential attack underway ...

# Multilevel Policies

- Put programs at the lowest security level, all subjects at higher levels
  - By \*-property, nothing can write to those programs
  - By ss-property, anything can read (and execute) those programs
- Example: DG/UX system
  - All executables in “virus protection region” below user and administrative regions

# Detect Alteration of Files

- Compute manipulation detection code (MDC) to generate signature block for each file, and save it
- Later, recompute MDC and compare to stored MDC
  - If different, file has changed
- Example: tripwire
  - Signature consists of file attributes, cryptographic checksums chosen from among MD4, MD5, HAVAL, SHS, CRC-16, CRC-32, etc.)

# Behavior-Blocking Software

- integrated with host O/S
- monitors program behavior in real-time
  - eg file access, disk format, executable mods, system settings changes, network access
- for possibly malicious actions
  - if detected can block, terminate, or seek ok
- has advantage over scanners
- but malicious code runs before detection

# Antivirus Programs

- Look for specific sequences of bytes (called “virus signature” in file
  - If found, warn user and/or disinfect file
- Each agent must look for known set of viruses
- Cannot deal with viruses not yet analyzed
  - Due in part to undecidability of whether a generic program is a virus

# Virus Countermeasures

- best countermeasure is prevention
- but in general not possible
- hence need to do one or more of:
  - **detection** - of viruses in infected system
  - **identification** - of specific infecting virus
  - **removal** - restoring system to clean state

# Anti-Virus Software

- **first-generation**
  - scanner uses virus signature to identify virus
  - or change in length of programs
- **second-generation**
  - uses heuristic rules to spot viral infection
  - or uses crypto hash of program to spot changes
- **third-generation**
  - memory-resident programs identify virus by actions
- **fourth-generation**
  - packages with a variety of antivirus techniques
  - eg scanning & activity traps, access-controls
- arms race continues

# Advanced Anti-Virus Techniques

- generic decryption
  - use CPU simulator to check program signature & behavior before actually running it
- digital immune system (IBM)
  - general purpose emulation & virus detection
  - any virus entering org is captured, analyzed, detection/shielding created for it, removed

# N-Version Programming

- Implement several different versions of algorithm
- Run them concurrently
  - Check intermediate results periodically
  - If disagreement, majority wins
- Assumptions
  - Majority of programs not infected
  - Underlying operating system secure
  - Different algorithms with enough equal intermediate results may be infeasible
    - Especially for malicious logic, where you would check file accesses

# Proof-Carrying Code

- Code consumer (user) specifies safety requirement
- Code producer (author) generates proof code meets this requirement
  - Proof integrated with executable code
  - Changing the code invalidates proof
- Binary (code + proof) delivered to consumer
- Consumer validates proof
- Example statistics on Berkeley Packet Filter: proofs 300–900 bytes, validated in 0.3 –1.3 ms
  - Startup cost higher, runtime cost considerably shorter