

# Malware

ECS 153 Spring Quarter 2021

Module 19

# Overview

- Defining malware (also called *malicious logic*)
- Types
  - Trojan horses
  - Computer viruses and worms
  - Other types
- Theory: arbitrary program being a virus undecidable?
- Defenses
  - Properties of malicious logic
  - Trust

# Malware

- Set of instructions that cause site security policy to be violated

# Example

- Shell script on a UNIX system:

```
cp /bin/sh /tmp/.xyzzy  
chmod u+s,o+x /tmp/.xyzzy  
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: Gemini

- Designed for Android cell phones
- Placed in several Android apps on Android markets, forums
- When app was run:
  - Gemini installed itself, using several techniques to make it hard to find
  - Then it connected to a remote command and control server, waited for commands
  - Commands it could execute included delete SMS messages; send SMS messages to remote server; dump contact list; dump list of apps

# Rootkits

- Trojan horse corrupting system to carry out covert action without detection
- Earliest ones installed back doors so attackers could enter systems, then corrupted system programs to hide entry and actions
  - Program to list directory contents altered to not include certain files
  - Network status program altered to hide connections from specific hosts

# Example: Linux Rootkit IV

- Replaced system programs that might reveal its presence
  - *ls, find, du* for file system; *ps, top, lsof, killall* for processes; *crontab* to hide rootkit jobs
  - *login* and others to allow attacker to log in, acquire superuser privileges (and it suppressed any logging)
  - *netstat, ifconfig* to hide presence of attacker
  - *tcpd, syslogd* to inhibit logging
- Added back doors so attackers could log in unnoticed
- Also added network sniffers to gather user names, passwords
- Similar rootkits existed for other systems



# Defenses

- Use non-standard programs to obtain the same information that standard ones should; then compare
  - *ls* lists contents of directory
  - *dirdump*, a program to read directory entries, was non-standard
    - Compare output to that of *ls*; if they differ, *ls* probably compromised
- Look for specific strings in executables
  - Programs to do this analysis usually not rigged, but easy enough to write your own
- Look for changes using cryptographically strong checksums
- These worked because they bypassed system programs, using system calls directly

# Next Step: Alter the Kernel

- Rootkits then altered system calls using kernel-loadable modules
  - Thereby eliminating the effectiveness of the earlier defenses
- Example: Knark modifies entries in system call table to involve versions in new kernel-loadable module; these hide presence of Knark
  - Defense: compare system call table in kernel with copy stored at boot time
- Example: SuckKIT changes variable in kernel that points to system call table so it points to a modified table, defeating the Knark defense
- Example: adore-ng modifies virtual file system layer to hide files with rootkit's UID or GID; manipulates /proc and other pseudofiles to control what process monitoring programs report
  - Takes advantage of the ability to access OS entities like processes through file system

# Oops ...

- Sony BMG developed rootkit to implement DRM on a music CDs
  - Only worked on Windows systems; users had to install a proprietary program to play the music
  - Also installed software that altered functions in Windows OS to prevent playing music using other programs
  - This software concealed itself by altering kernel not to list any files or folders beginning with “\$sys\$” and storing its software in such a folder
  - On boot, software contacted Sony to get advertisements to display when music was played
  - Once made public, attackers created Trojan horses with names beginning with “\$sys\$ (like “\$sys\$drv.exe”)
- Result: lawsuits, flood of bad publicity, and recall of all such CDs

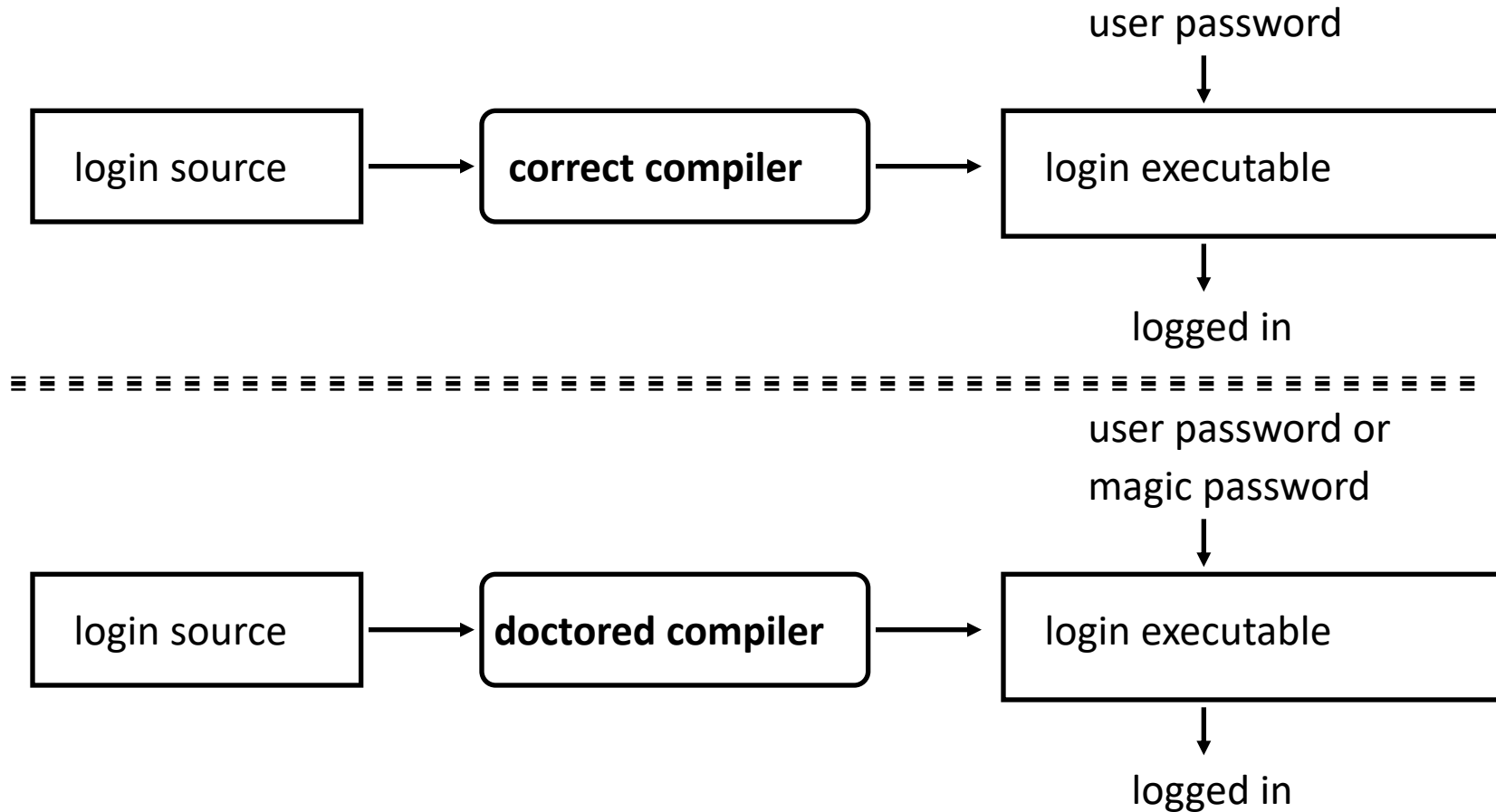
# Replicating Trojan Horse

- 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

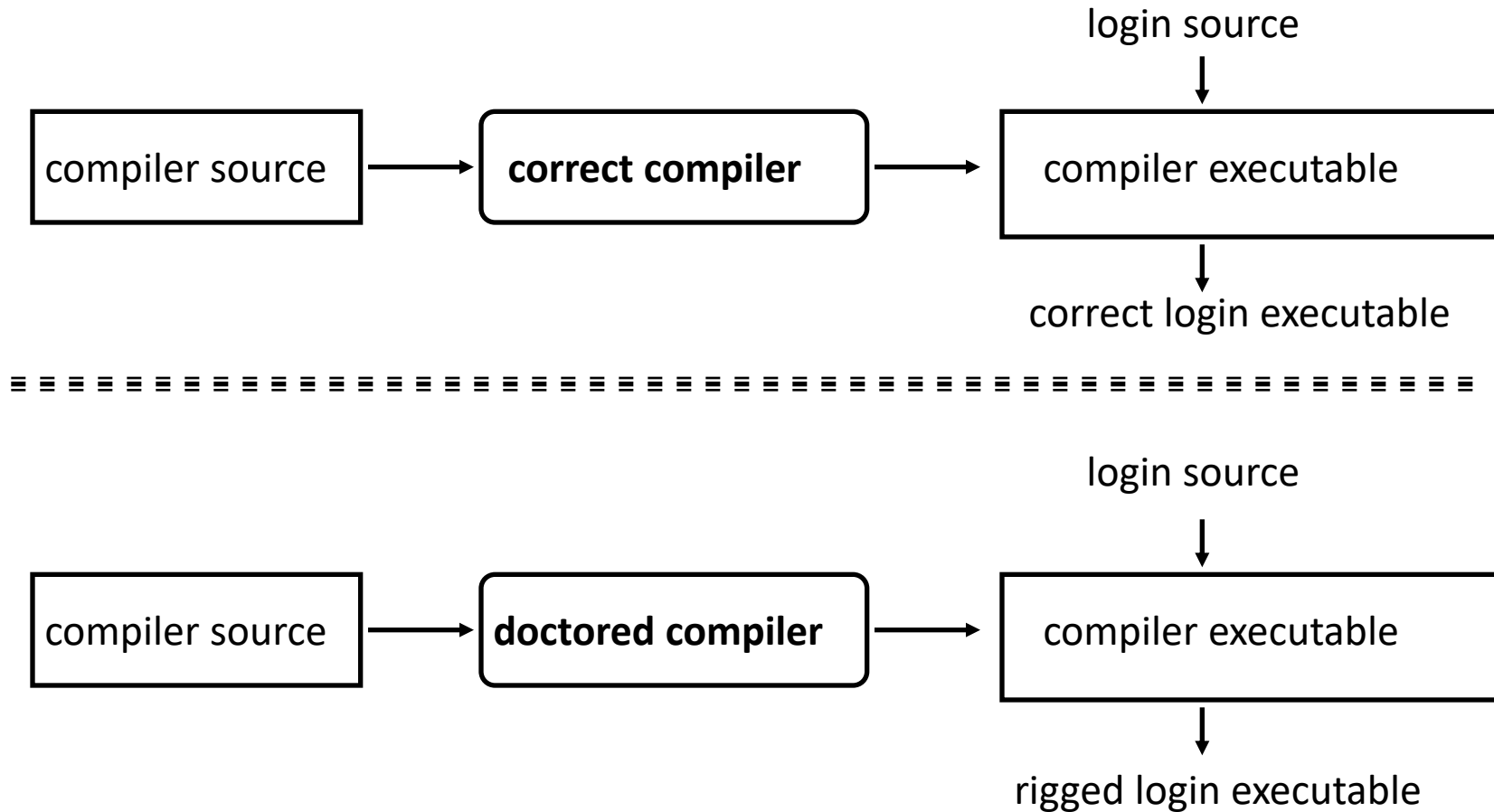
# Thompson's Compiler

- 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



# 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*

endvirus:

# Trojan Horse Or Not?

- Yes
  - Overt action = infected program's actions
  - Covert action = virus' actions (infect, execute)
- No
  - Overt purpose = virus' actions (infect, execute)
  - Covert purpose = none
- Semantic, philosophical differences
  - Defenses against Trojan horse also inhibit computer viruses

# History

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

# Cohen's Experiments

- UNIX systems: goal was to get superuser privileges
  - Max time 60m, min time 5m, average 30m
  - Virus small, so no degrading of response time
  - Virus tagged, so it could be removed quickly
- UNIVAC 1108 system: goal was to spread
  - Implemented simple security property of Bell-LaPadula
  - As writing not inhibited (no \*-property enforcement), viruses spread easily

# First Reports of Viruses in the Wild

- Brain (Pakistani) 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

# Infection Vectors

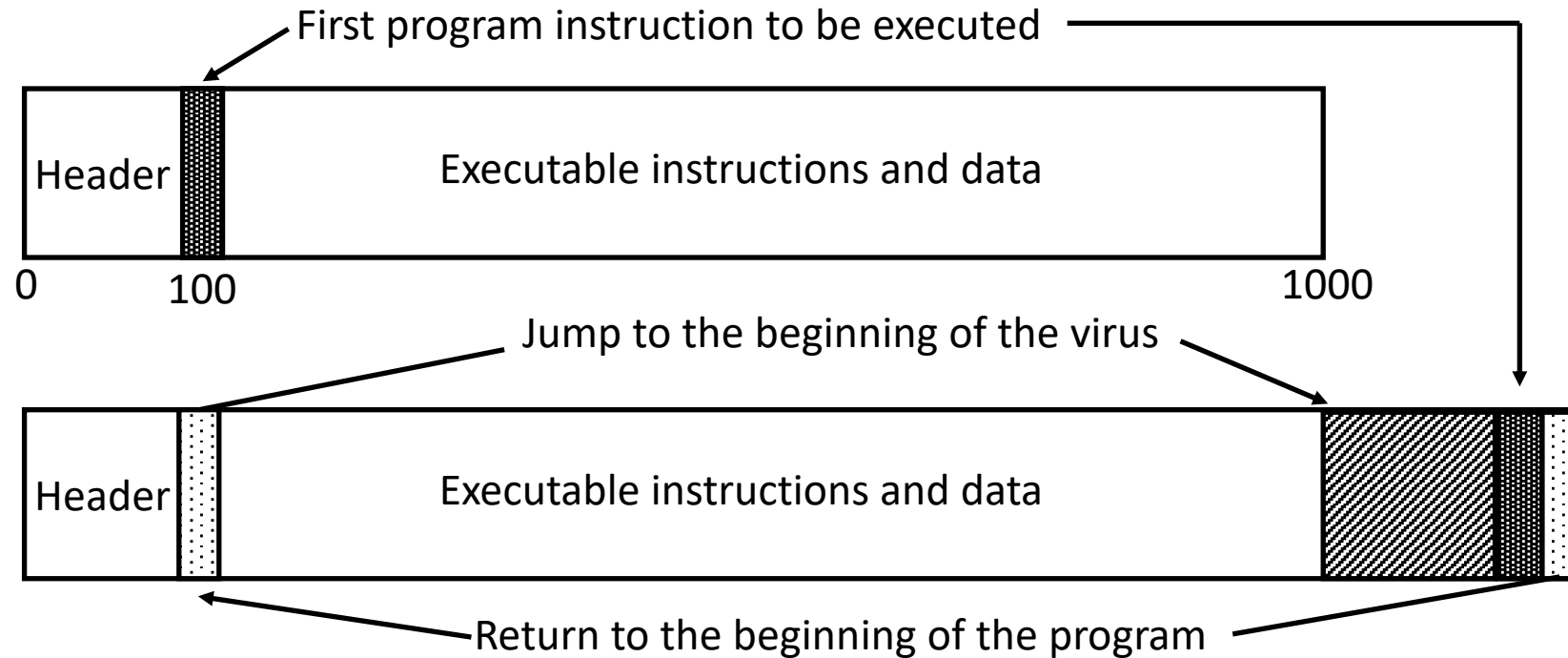
- Boot sector infectors
- Executable infectors
- Data infectors
- These are not mutually exclusive; some viruses do two or three of these



# 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; if no free space, doesn't infect but if any free space, it infects, possibly overwriting used disk space
    - If there, jumps to vector at 6DH

# Executable Infectors



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

# Executable Infectors (*con't*)

- Jerusalem (Israeli) 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 COMMAND.COM
    - Otherwise, infect or delete
  - Error: doesn't set signature when .EXE executes
    - So .EXE files continually reinfected

# 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
    - Used a mail program that most Macintosh users didn’t use, so this was rare for Macintosh users

# Multipartite Viruses

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

# Concealment

- Terminate and stay resident (TSR)
- Stealth
- Encryption
- Polymorphism
- Metamorphism

# TSR Viruses

- A virus that stays active in memory after the application (or bootstrapping, or disk mounting) is completed
  - Non-TSR viruses only execute when host application executes
- Examples: Brain, Jerusalem viruses
  - Stay in memory after program or disk mount is completed

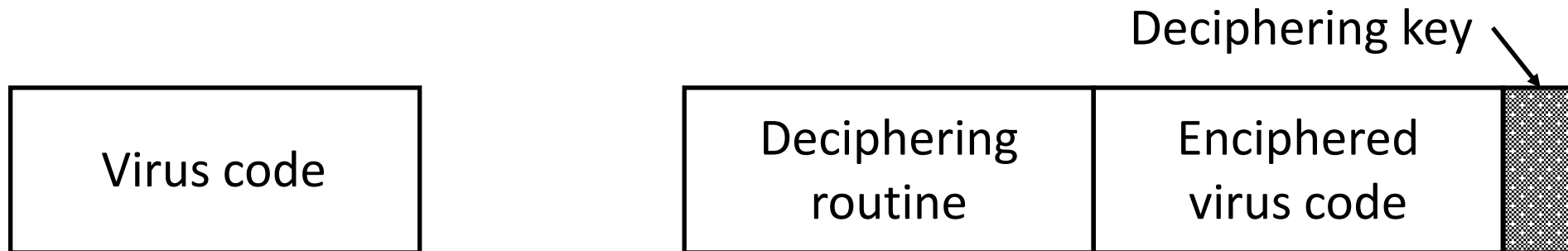


# Stealth Viruses

- A virus that conceals infection of files
- Example: IDF (also called Stealth or 4096) 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)
- After decipherment, same virus loaded into memory
  - Virus is encrypted; decryption routine is obscured (polymorphicized?)

# 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

# Metamorphic

- Like polymorphic, but virus itself is also obscured
  - So two instances of virus would look different when loaded into memory
- When decrypted, virus may have:
  - Two completely different implementations
  - Two completely different algorithms producing same result

# Example

- W95/Zmist virus distributes pitself throughout code being infected
- On finding file to infect:
  - $p = 0.1$ : insert jump instructions between each set of non-jump instructions
  - $p = 0.1$ : infect file with unencrypted copy of Zmist
  - $p = 0.8$ : if file has section with initialized data that is writeable, infect file with polymorphic encrypted version of Zmist; otherwise, infect file with unencrypted copy of Zmist
    - In first case, virus expands that section, inserts virus code as it is decrypted, and executes that code; decrypting code preserves registers so they can be restored
- On execution, allocates memory to put virus engine in; that creates new instance of (transformed) virus

# 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: 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

# 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

# Computer Worm Structure

- *Target Selection*: worm determines which systems to spread to
- *Propagation*: worm attempts to infect chosen targets
- *Execution*: worm carries out action after it becomes resident on a target
  - This phase may be empty

# Example: Internet Worm

- Target selection: chose targets from lists of trusted hosts, and hosts trusted by users whose passwords had been guessed
- Propagation: tried to exploit 4 vulnerabilities
  - *sendmail* (SMTP server) in debug mode
  - *fingerd* (information server) buffer overflow attack
  - used guessed passwords
  - tried to exploit trust relationships
- Execution: took actions to:
  - Concealed its presence
  - Prevent reinfection
  - tried to guess passwords on local system (to be used in target selection)

# Stuxnet

- Found in 2010, targeted Siemens centrifuges used in process to enrich uranium
  - Compromised Windows software first, then the PLC in centrifuges
    - Very sophisticated evasion, exploits, and use of first PLC rootkit
  - Spun them at nonstandard speeds so they tore apart
- Entered system via infected USB stick with a Trojan horse
  - Looked on local network for Windows-based systems to infect; if found, infected no more than 3
- On system, checked to see if it was part of a specific industrial control system
  - No: did nothing
  - Ye: acted

# Stuxnet (*con't*)

- Tried to download most current version of itself
- Exploited vulnerabilities in infected system's PLC to take control of attached centrifuges
  - Also corrupted information sent to the controllers so they would not detect anything was wrong until centrifuges went out of control
- Believed developed by one or more nation-states due to its complexity, sophistication
- Other equally sophisticated worms found since then
  - Flame: spread in ways similar to Stuxnet, but only gathers information from microphones, keystrokes, network traffic, and so forth for the attackers to retrieve

# Importance of Stuxnet

- Earlier research showed physical systems vulnerable to attacks from connected computers
- Stuxnet showed these attacks can be launched over the Internet

# Bots

- *bot*: malware that carries out some action in co-ordination with other bots
- *botmaster*: attacker controlling the bots on one or more systems
- *command and control (C&C) server, mothership*: system(s) the attacker uses to control the bots
- *C&C channels*: communication paths used to communicate with bots
  - Distinguishing characteristic of bot is the use of this channel
  - Can be triggered, updated over this
- *botnet*: a collection of bots



# Life Cycle of a Bot in a Botnet

1. Bot infects system
2. Bot checks for a network connection, looks for either C&C server or another bot it can communicate with
3. Bot gets commands sent by C&C server or other bot
  - These may include adding components to add to what the bot can do
4. Bot executes these commands
  - May send results to somewhere else

# Organization of a Botnet

- *Centralized*; each bot communicates directly with C&C server
  - Potential problem: C&C server can become a bottleneck
- *Hierarchical*: C&C server communicates with set of bots, which in turn act as C&C servers for other bots, in a hierarchy
  - Good for controlling large botnets such as Torpig (over 180,000 bots) and Mirai (estimated to have 493,000 bots)
- *Peer-to-peer*: no single C&C server; bots act as peers, and botnet is a peer-to-peer network
  - High latency; to join the botnet, a bot scans addresses until it finds another bot, then forwards message to it

# Example: GTBot (Centralized)

- One of the earliest bots; used IRC channel as C&C channel
- Get it onto a Windows system running mIRC client
  - Installed scripts to monitor the channel looking for specific keywords
  - Also program to hide the bot, and possibly other programs to propagate the bot to other systems and install servers on them
- One instance (Backdoor.IRC.Aladinz) installs and hides itself, then notifies attacker (via IRC channel) it is resident
  - This has lots of tools to launch attacks

# Example: Torpig (Hierarchical)

- Based on Mebroot rootkit
- Bot installed itself so it is executed at boot, *before* the operating system loaded
- Then contacted Mebroot C&C server to get modules to add to bot
  - Compromises sensitive data like passwords
  - Data sent to Torpig C&C server
- Server acknowledges upload
  - Can also send file with addresses of backup Torpig C&C servers, and how often the bot should contact the server

# Example: Trojan.Peacomm (Peer-to-Peer)

- Bot gets on Windows system
- Uses peer-to-peer protocol to connect to peers
  - Addresses of at least 100 peers stored in bot
- Looks for a value encoding a UTL pointing to another component
  - Downloads and executes this component
  - Many possible component, including one for spamming, one for reaping email addresses for spam, one to launch a distributed denial of service attack

# Content Delivery Networks and IP Flux

- Content delivery networks (Netflix, Amazon, etc.) have many servers
  - These are invisible to the client
  - When DNS gets target system, it returns IP address to client
- As these networks grew, needed way to prevent any single server from being overloaded
- Mechanism is to change IP address associated with a particular host name over a very short period of time (called *IP flux*)
  - So, while traffic appears to go to one particular system, it is actually sent to whichever server has the address currently

# Applying This to Botnets

- Goal: make it harder to locate bots
  - Especially C&C servers
- Approach: associate list of IP addresses with host name
- *IP flux botnet*: change binding between host name (in bot) and IP address, going down the list (*flux*)
  - *Single flux botnet*: list of IP addresses is large; host name is registered to one of these addresses, and after a short time deregistered and reregistered using a different IP address
  - *Double flux botnet*: do the same with IP addresses of the DNS servers used to look up the IP addresses
- Also called *fast flux botnets*

# Example: Flame (Fast Flux Botnet)

- Bot gathers information from infected system, network
  - This is then retrieved over C&C channel to Flame server
  - Server can also send new modules for bot
- When Flame installed:
  - Check for network by trying to connect to well-known servers; assume successful
  - Flame had 5 domains initially for the Flame server; Flame server could add more to this list
    - Communication over SSL; Flame's certificate was self-signed
  - Preliminary analysis: more than 50 host names and more than 15 IP addresses related to C&C messages; later raised number of C&C hosts to around 100



# Variation: Domain Flux

- Instead of holding host name fixed and changing the associated IP address, change host name and keep associated IP address fixed
  - Called *domain flux*
- Advantage: host name associated with C&C server for a short period of time
  - With IP flux, finding host name identifies C&C server
- Example: Torpig
  - Compute a host name that is fixed for current week number and year; append “.com”, “.net”, “.biz” suffixes and probe for each
  - If no luck, generate name based on current day, append suffixes, probe for each
  - If no luck, go to fixed list of host names

# Rabbit, Bacterium

- 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

# Adware

- Trojan horse that gathers information for marketing purposes and displays advertisements
  - Often selects ads to display based on gathered information
- Believed to have originated with a company announcing it would make its software available for free, because it would pop up window advertising company
  - Benign as user had to opt in
  - Spread through distribution of program only

# Types of Behavior

- *Low severity behavior*: just display ads, don't transmit information
- *Medium severity behavior*: transmits information deemed low risk, such as location information, and may display ads based on this
- *High severity behavior*: transmits personal information, and displays ads tailored to devices, people with those characteristic
  - Typically very aggressive (annoying)
  - Sometimes called *malware*

# Getting Adware On a System

- Put on a web site user visits
  - Put it in a banner enticing the user to click on it; this installs the adware
  - Page may require user to install software to view parts of web site; software contains adware
  - If page refreshes automatically, it may direct browser to run an executable
    - Usually browser notifies user via a dialog box that may require a click; on click, program runs and installs adware
  - Some browser plug-ins download, execute files automatically; there may be no indication of this
    - Called *drive-by downloading*

# Getting Adware on a System

- Put into software that user downloads
  - Very common with mobile apps
- Problem: app asks for permission to carry out its tasks
  - Some may be unnecessary; often hard for users to minimize permissions set
  - Thus app may have access to camera, microphone, and may be able to make calls without going through dialing interface — and user does not realize this
- Example: survey of 900 Android apps
  - 323 had unnecessary permissions

# Economic Motives

- Used to target ads that use is most likely to respond to
- Purveyors get money for every ad displayed or clicked on
  - Web site owners display ads on their sites
  - Developers put adware libraries in their apps
  - Others take apps, modify them to include adware, and put them on unauthorized app stores



# Spyware

- Trojan horse that records information about the use of a computer for a third party
  - Usually results in compromise of confidential information like keystrokes, passwords, credit card numbers, etc.
  - Information can be stored for retrieval or sent to third party
- Put on a system the way any other malware gets onto system

# Example: Pegasus

- Designed for Apple's iPhone, attacker sends URL to victim who clicks on it, triggering attack that tries to gain control of iPhone
- First sends HTML file exploiting vulnerability in WebKit
  - Basis for Safari and other browsers
- This downloads software to gain control of iPhone
  - Software enciphered with different keys for each download
  - Includes a loader for the next stage
- Loader downloads dynamic load libraries, daemons, other software and installs Pegasus
  - If iPhone has previously been jailbroken, removes all access to the iPhone provided by the earlier break

# Example: Response to Pegasus

- Apple developed patches for the vulnerabilities exploited
  - Deployed them in update to iPhone's operating system, iOS
- Discovered when human rights activist received text messages with a suspicious link
  - Sent messages to Citizens Lab
  - Citizens Lab recognized links were associated with a manufacturer of spyware for government surveillance
  - Lookout carried out technical analysis

# Ransomware

- Malware inhibiting use of computer, resources until a ransom is paid
  - Ransom is usually monetary, and must be paid through some anonymous mechanism (BitCoin is popular)
- PC CYBORG (1989) altered AUTOEXEC.BAT to count number of times system was booted; on 90<sup>th</sup>, names of all files on main drive (C:) enciphered and directories hidden
  - User told to send fee to post office box to recover the system
- CryptoLocker (2013) encrypted files and gave victim 100 hours to pay ransom; if not, encryption keys destroyed
  - Used evasive techniques to make tracking more difficult
  - Spread via email as attachments

# Example Protocol

Goal: Angie wants to extort money from Xavier

- Angie generates asymmetric key pair; embeds public key in malware
  - She retains private key
- Malware infects Xavier's system
  - Generates symmetric key and uses that to encipher target data
  - Enciphers symmetric key with public key, erases all instances of symmetric key
  - Xavier sees message saying he needs to do something for Angie (usually send money); he does so and includes the encrypted symmetric key
- Angie then deciphers encrypted symmetric key with her private key, returns it to Xavier

# Phishing

- Act of impersonating legitimate entity in order to obtain information such as passwords or credit card information without authorization
  - Usually a web site associated with a business
- Usual approach: craft a web site that looks like the legitimate one
  - Send out lots of email trying to persuade people to go to that web site
  - Copy their login, password, and other information for later use
- More vicious attack: fake web site passes data on to real web site, and sends replies back to victim
  - Man-in-the-middle attack

# Example

- Heidi banks at MegaBank, with URL of <https://www.megabank.com>
- She receives a letter saying she needs to check her account for possible fraudulent activity
- Email includes link
  - Link is visible as [www.megabank.com](http://www.megabank.com)
  - But link actually connects to <https://www.megabank.crookery.com>
- Attacker records name, password, then give error
  - If very clever, client is redirected to actual bank's home page

# Spearphishing

- Phishing attack tailored for particular user
- Used to attack specific (types of) users to obtain information
- Example: some employees of a major cybersecurity company received email called “2011 Recruitment Plan”
  - They opened an attacked spreadsheet
  - This exploited a vulnerability in a supporting program to install a backdoor so attackers could control system remotely
  - Attackers used this as a springboard to compromise other systems in the company’s network, and ultimately stole sensitive information
  - Embarrassment, financial costs of recovery large



# Theory of Viruses

- **Theorem:** It is undecidable whether an arbitrary program contains malicious logic

# Defenses

- Scanning
- Distinguishing between data, instructions
- Containing
- Specifying behavior
- Limiting sharing
- Statistical analysis

# Scanning Defenses

- Malware alters memory contents or disk files
- Compute manipulation detection code (MDC) to generate signature block for data, and save it
- Later, recompute MDC and compare to stored MDC
  - If different, data has changed

# Example: *tripwire*

- File system scanner
- Initialization: it computes signature block for each file, saves it
  - Signature consists of file attributes, cryptographic checksums
  - System administrator selects what file attributes go into signature
- Checking file system: run *tripwire*
  - Regenerates file signatures
  - Compares them to stored file signatures and reports any differences

# Assumptions

- Files do not contain malicious logic when original signature block generated
- Pozzo & Grey: implement Biba's model on LOCUS to make assumption explicit
  - Credibility ratings assign trustworthiness numbers from 0 (untrusted) to  $n$  (signed, fully trusted)
  - Subjects have risk levels
    - Subjects can execute programs with credibility ratings  $\geq$  risk level
    - If credibility rating  $<$  risk level, must use special command to run program

# Antivirus Programs

- Look for specific “malware signatures”
  - If found, warn user and/or disinfect data
- At first, static sequences of bits, or patterns; now also includes patterns of behavior
- At first, derived manually; now usually done automatically
  - Manual derivation impractical due to number of malwares

# Example: Earlybird

- System for generating worm signatures based on worm increasing network traffic between hosts, and this traffic has many common substrings
- When a packet arrives, its contents hashed and destination port and protocol identifier appended; then check hash table (called *dispersion table*) to see if this content, port, and protocol have been seen
  - If yes, increment counters for source, destination addresses; if both exceed a threshold, content may be worm signature
  - If no, run through a multistage filter that applies 4 different hashes and checks for those hashes in different tables; count of entry with smallest count incremented; if all 4 counters exceed a second threshold, make entry in dispersion table
- Found several worms before antimalware vendors distributed signatures for them

# Behavioral Analysis

- Run suspected malware in a confined area, typically a sandbox, that simulates environment it will execute in
- Monitor it for some period of time
- Look for anything considered “bad”; if it occurs, flag this as malware



# Example: Panorama

- Loads suspected malware into a Windows system, which is itself loaded into Panorama and run
  - Files belonging to suspect program are marked
- Test engine sends “sensitive” information to trusted application on Windows
- Taint engine monitors flow of information around system
  - So when suspect program and trusted application run, behavior of information can be recorded in taint graphs
- Malware detection engine analyzes taint graphs for suspicious behavior
- Experimentally, Panorama tested against 42 malware samples, 56 benign samples; no false negatives, 3 false positives

# Evasion

Malware can try to ensure malicious activity not triggered in analysis environment

- Wait for a (relatively) long time
- Wait for a particular input or external event
- Identify malware is running in constrained environment
  - Check various descriptor tables
  - Run sequence of instructions that generate an exception if not in a virtual machine (in 2010, estimates found 2.13% of malware samples did this)

# 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
  - Key are assumption that certifying authority will *not* make mistakes and assumption that tools, supporting infrastructure used in certifying process are not corrupt

# Example: Duff and UNIX

- Observation: users with execute permission usually have read permission, too
  - So files with “execute” permission have type “executable”; those without it, type “data”
  - Executable files can be altered, but type immediately changed to “data”
    - Implemented by turning off execute permission
    - Certifier can change them back
      - So virus can spread only if run as certifier

# 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

# Karger's Scheme

- Base it on attribute of subject, object
- Interpose a knowledge-based subsystem to determine if requested file access reasonable
  - Sits between kernel and application
- Example: UNIX C compiler
  - Reads from files with names ending in “.c”, “.h”
  - Writes to files with names beginning with “/tmp/ctm” and assembly files with names ending in “.s”
- When subsystem invoked, if C compiler tries to write to “.c” file, request rejected

# Lai and Gray

- Implemented modified version of Karger's scheme on UNIX system
  - Allow programs to access (read or write) files named on command line
  - Prevent access to other files
- Two types of processes
  - Trusted: no access checks or restrictions
  - Untrusted: valid access list (VAL) controls access and is initialized to command line arguments plus any temporary files that the process creates

# File Access Requests

1. If file on VAL, use effective UID/GID of process to determine if access allowed
2. If access requested is read and file is world-readable, allow access
3. If process creating file, effective UID/GID controls allowing creation
  - Enter file into VAL as NNA (new non-argument); set permissions so no other process can read file
4. Ask user. If yes, effective UID/GID controls allowing access; if no, deny access



# Example

- Assembler invoked from compiler

```
as x.s /tmp/ctm2345
```

and creates temp file /tmp/as1111

- VAL is

```
x.s /tmp/ctm2345 /tmp/as1111
```

- Now Trojan horse tries to copy x.s to another file
  - On creation, file inaccessible to all except creating user so attacker cannot read it (rule 3)
  - If file created already and assembler tries to write to it, user is asked (rule 4), thereby revealing Trojan horse

# Trusted Programs

- No VALs applied here
  - UNIX command interpreters: *cs**h*, *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

# Trust

- Trust the user to take explicit actions to limit their process' protection domain sufficiently
  - That is, enforce least privilege correctly
- Trust mechanisms to describe programs' expected actions sufficiently for descriptions to be applied, and to handle commands without such descriptions properly
- Trust specific programs and kernel
  - Problem: these are usually the first programs malicious logic attack

# 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 so that:
  - On first call, unique file identifier (inode) saved
  - On second call, inode of named file compared to that of first call; if they differ, attack may be underway ...

# Specifications

- Treat infection, execution phases of malware as errors
- Example
  - Break programs into sequences of non-branching instructions
  - Checksum each sequence, encrypt it, store it
  - When program is run, processor recomputes checksums, and at each branch compares with precomputed value; if they differ, an error has occurred

# Inhibit Sharing

- Use separation implicit in integrity policies
- Example: LOCK keeps single copy of shared procedure in memory
  - Master directory associates unique owner with each procedure, and with each user a list of other users the first trusts
  - Before executing any procedure, system checks that user executing procedure trusts procedure owner



# 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: Trusted Solaris system
  - All executables, trusted data stored below user region, so user applications cannot alter them

# Detecting Statistical Changes

- Example: application had 3 programmers working on it, but statistical analysis shows code from a fourth person—may be from a Trojan horse or virus!
- Other attributes: more conditionals than in original; look for identical sequences of bytes not common to any library routine; increases in file size, frequency of writing to executables, etc.
  - Denning: use intrusion detection system to detect these