

Lecture 21

November 12, 2025

ECS 235A, Computer and Information Security

Administrative Stuff

- Homework 4, Extra Credit 4 posted
 - Due on Friday, November 21
- Office hour today is moved to 1:00pm Thursday
 - On Thursday, the 12:10pm–1:00pm office hour will be held in 2203 Watershed Sciences, as usual
 - On Thursday, a second office hour, from 1:10pm–2:00pm, will be held over the usual Zoom link

Debate about Penetration Testing

- How valid are these tests?
 - Not a substitute for good, thorough specification, rigorous design, careful and correct implementation, meticulous testing
 - Very valuable *a posteriori* testing technique
 - Ideally unnecessary, but in practice very necessary
- Finds errors introduced due to interactions with users, environment
 - Especially errors from incorrect maintenance and operation
 - Examines system, site through eyes of attacker

Problems

- Flaw Hypothesis Methodology depends on caliber of testers to hypothesize and generalize flaws
- Flaw Hypothesis Methodology does not provide a way to examine system systematically
 - Vulnerability classification schemes help here

Malware

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

Example

- 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: 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
- App was the Trojan horse
 - Although the term was used for Gemini, too

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: SuckIT 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

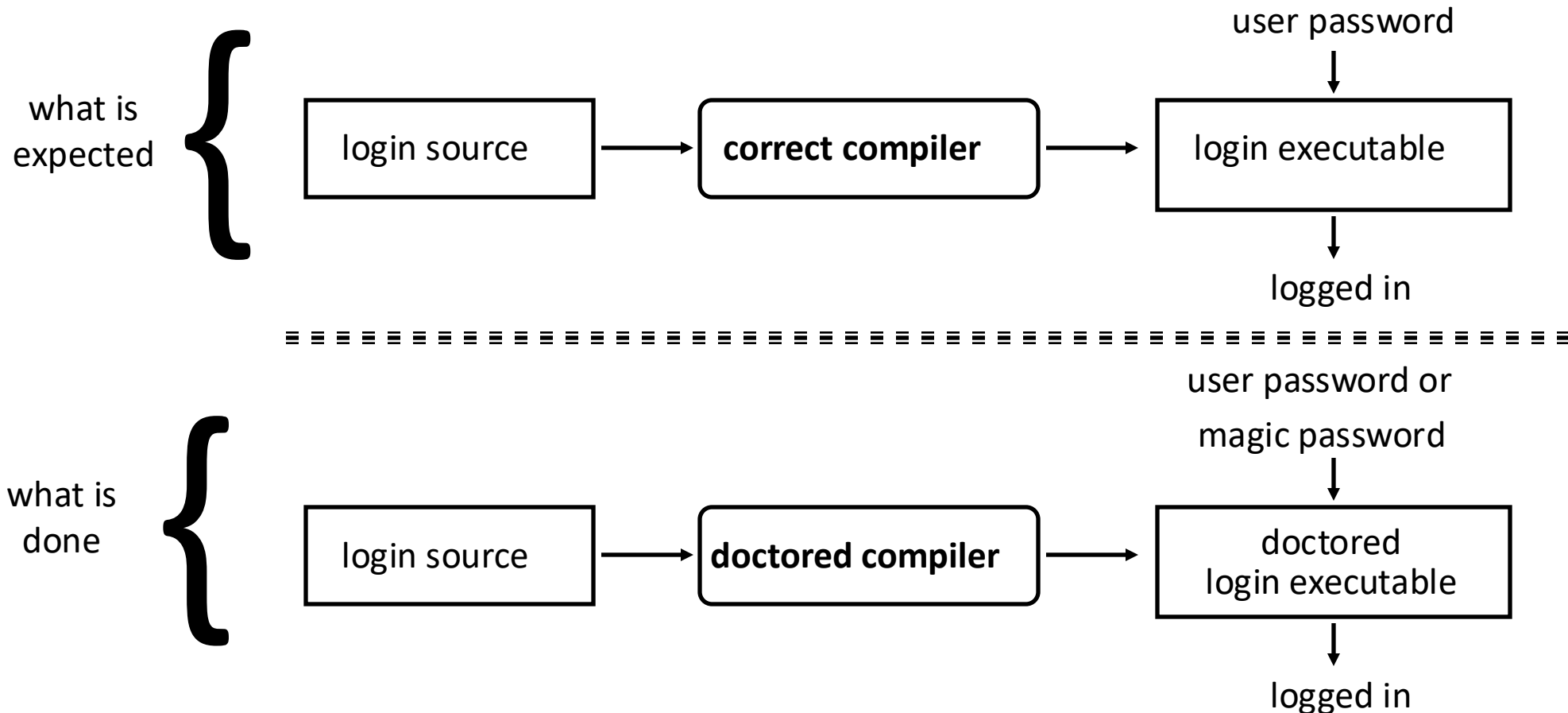
Replicating Trojan Horse

- 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 implemented this
 - See his lecture given upon receiving the Turing Award

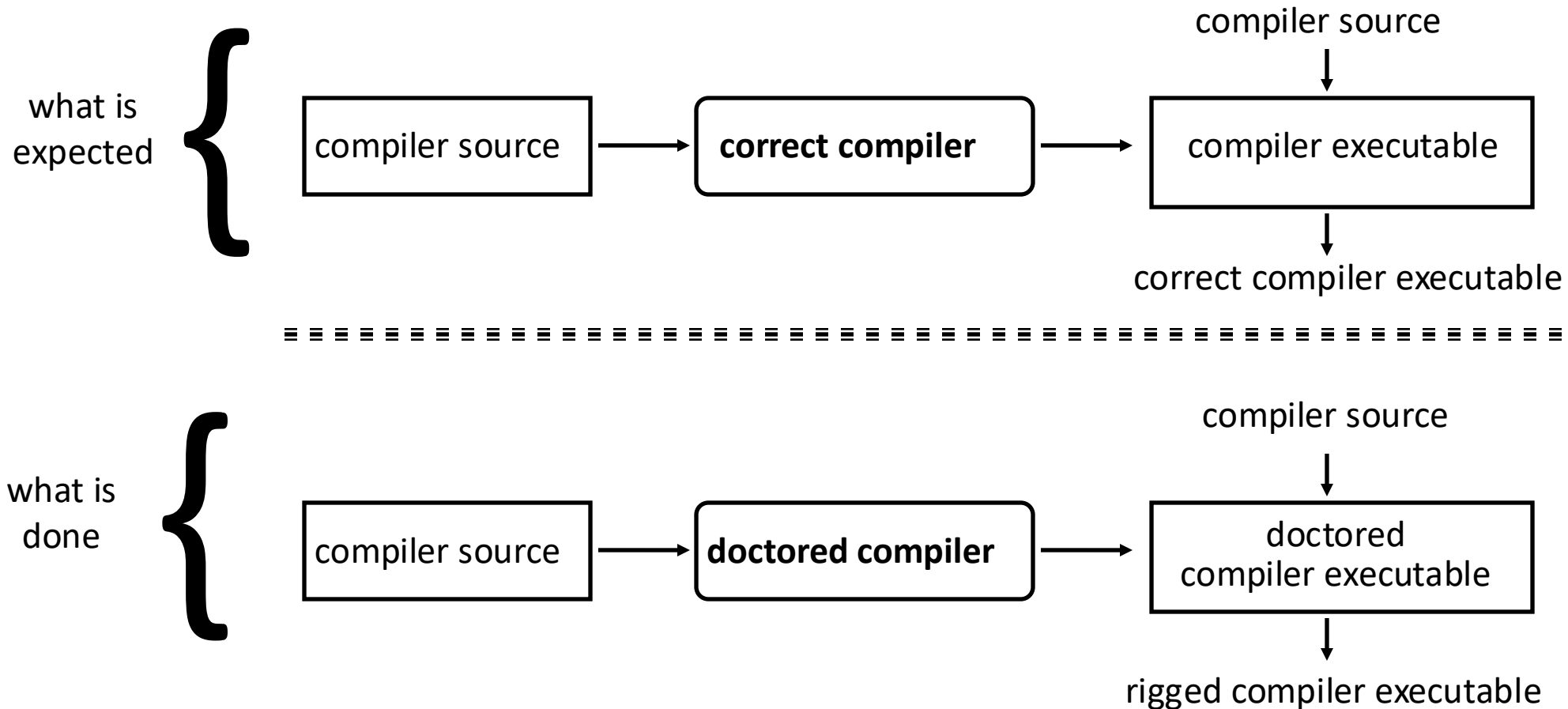
Thompson's Compiler

- Modify the compiler so that when it compiles *login.c*, *login* executable 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 (Len 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

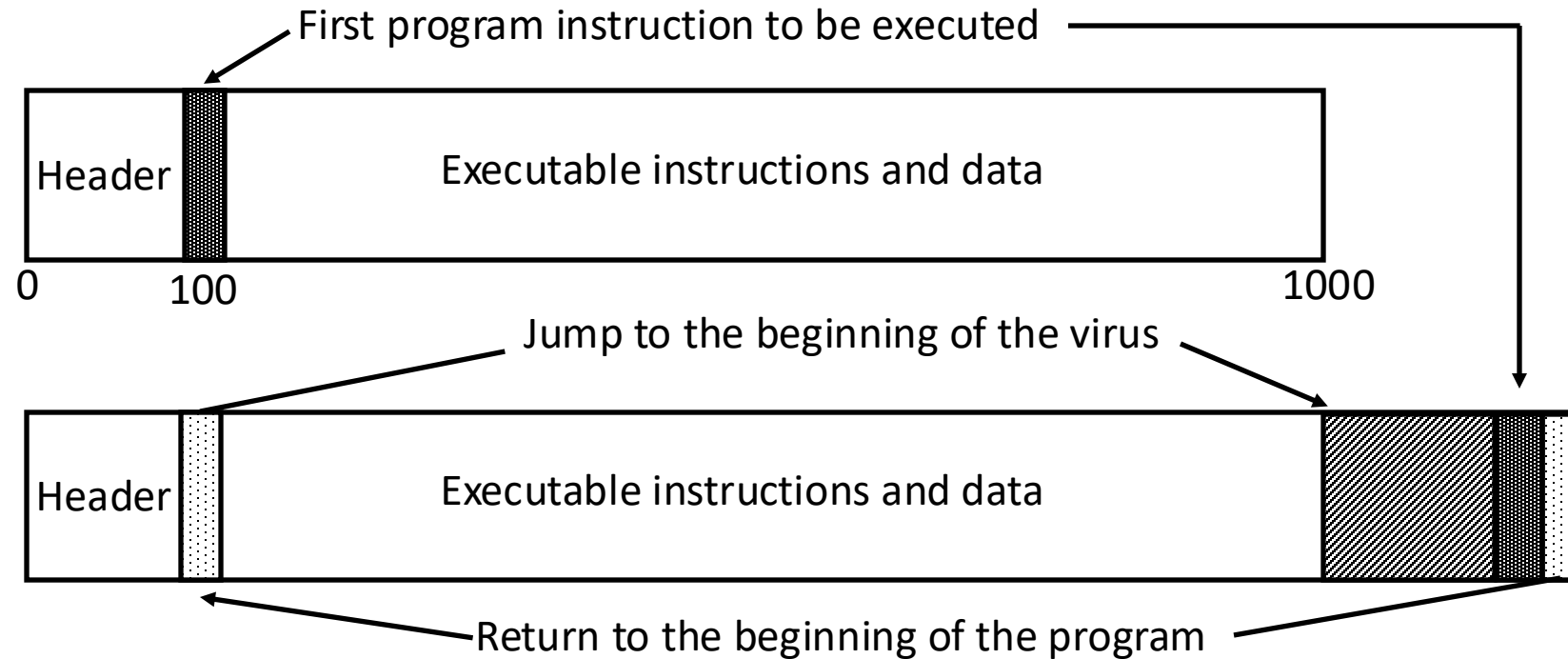
Infection Vectors

- 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 disk inserted, 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 opened infected file
 - Installed itself as “open” macro and copied itself into Normal template
 - This way, infected any files that are opened in future
 - Invoked mail program, sent 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

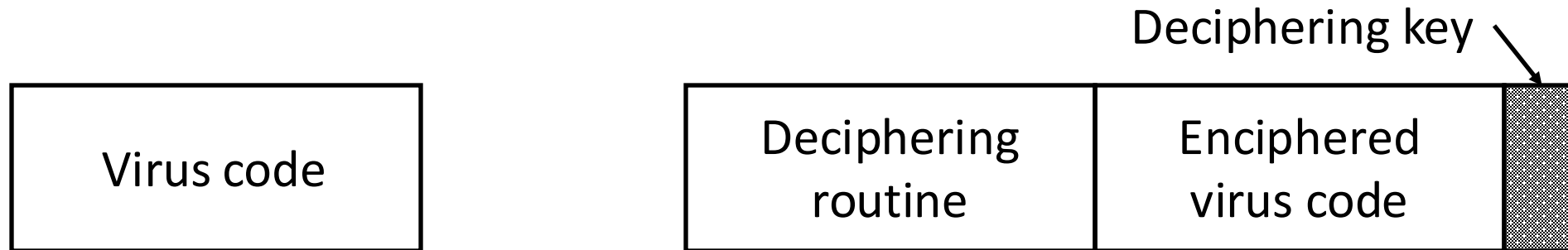
- A virus that stays active in memory after the application (or bootstrapping, or disk mounting) is completed
 - Non-TSR viruses only execute when host application executes
- Examples: Brain, Jerusalem viruses
 - Stay in memory after program or disk mount is completed

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 itself 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 Worm

- A program that copies itself from one computer to another
- Originated as a tool for 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