

# CSC 490 Special Topics

## Computer and Network Security

### Chapter 19: Malicious Logic

**Dr. Xiao Qin**

*Auburn University*

*<http://www.eng.auburn.edu/~xqin>*

*[xqin@auburn.edu](mailto:xqin@auburn.edu)*

# Malicious Logic

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

# 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: NetBus

- Designed for Windows 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

# 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

# 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

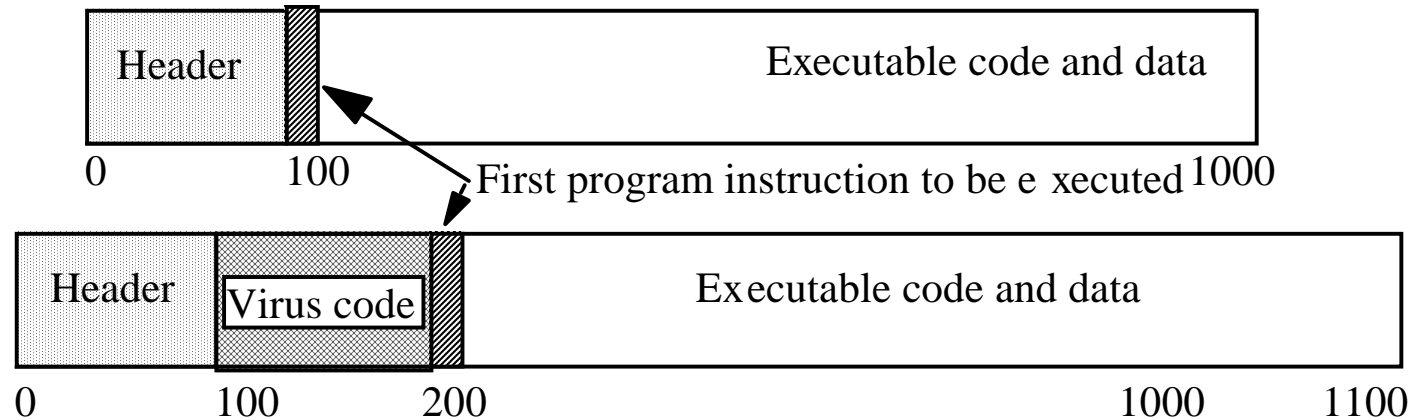
# Types of Viruses

- 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 (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 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

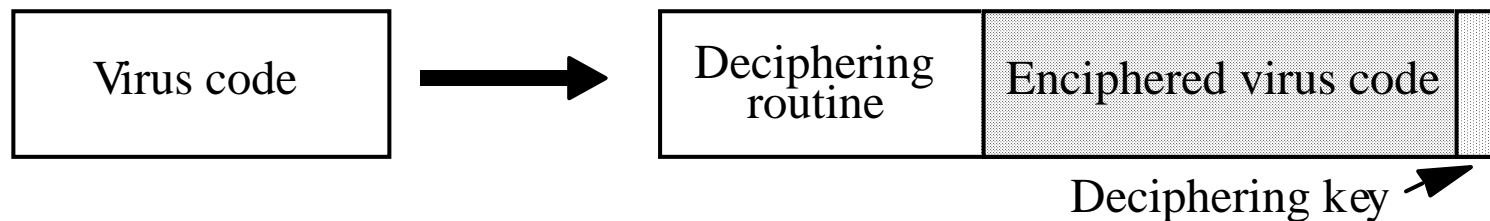
- A virus that stays active in memory after the application (or bootstrapping, or disk mounting) is completed
  - TSR is “**Terminate and Stay Resident**”
- Examples: Brain, Jerusalem viruses
  - Stay in memory after program or disk mount is completed

# 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



# 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)

# 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: 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

# Example: LOCK

- Logical Coprocessor Kernel
  - Designed to be certified at TCSEC A1 level
- **Compiled programs are type “data”**
  - Sequence of specific, auditable events required to change type to “executable”
- **Cannot modify “executable” objects**
  - So viruses can't insert themselves into programs (no infection phase)

# 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

# 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

# Key Points

- A perplexing problem
  - How do you tell what the user asked for is *not* what the user intended?
- Strong typing leads to separating data, instructions
- File scanners most popular anti-virus agents
  - Must be updated as new viruses come out