Chapter 19: Malicious Logic

Definition

- Malicious software (or simply malware) is software whose existence or execution has negative or unintended consequences

- Examples
  - Backdoors
  - Logic bombs
  - Virus
  - Trojan Horses
  - Worms
  - Botnets
  - Spyware
  - …

- Various types of payloads, ranging from annoyance to crime

Malware Classification

- Propagation
  - Virus: human-assisted propagation (e.g., open email attachment)
  - Worm: automatic propagation without human assistance
- Concealment
  - Rootkit: modifies operating system to hide its existence
  - Trojan: provides desirable functionality but hides malicious operation

Insider Attacks

- An **insider attack** is a security breach that is caused or facilitated by someone who is a part of the very organization that controls or builds the asset that should be protected.

- In the case of malware, an insider attack refers to a security hole that is created in a software system by one of its programmers.

Backdoors

- A **backdoor,** which is also sometimes called a **trapdoor,** is a hidden feature or command in a program that allows a user to perform actions he or she would not normally be allowed to do.
- When used in a normal way, this program performs completely as expected and advertised.
- But if the hidden feature is activated, the program does something unexpected, often in violation of security policies, such as performing a privilege escalation.
- Reasons for creating them
  - For fun: **Easter Eggs** (harmless undocumented features) in DVDs and software
  - For debugging purpose (e.g. a biometric authentication system)
  - Maliciously introduced (e.g., digital entry system for a bank vault); intentional bug

Logic Bombs

- A **logic bomb** is a program that performs a malicious action as a result of a certain logic condition.
- The classic example of a logic bomb is a programmer coding up the software for the payroll system who puts in code that makes the program crash should it ever process two consecutive payrolls without paying him.
- Another classic example combines a logic bomb with a backdoor, where a programmer puts in a logic bomb (can be disabled via the backdoor) that will crash the program on a certain date.

The Omega Engineering Logic Bomb

- An example of a logic bomb that was actually triggered and caused damage
- Programmer **Tim Lloyd** was convicted of using logic bomb on his former employer, Omega Engineering Corporation
- On July 31, 1996, a logic bomb was triggered on the server for Omega Engineering's manufacturing operations, which ultimately cost the company **millions of dollars** in damages and led to it laying off many of its employees

The Omega Bomb Code

- The Logic Behind the Omega Engineering Time Bomb included the following strings:
- 7/30/96
  - Event that triggered the bomb
- F:
  - Focused attention to volume F, which had critical files
- F:\LOGIN\LOGIN 12345
  - Login a fictitious user, 12345 (the back door)
- CD \PUBLIC
  - Moves to the public folder of programs
- FIX.EXE /Y F:*.*
  - Run a program, called FIX, which actually deletes everything
- PURGE F:\/ALL
  - Prevent recovery of the deleted files

Defenses against Insider Attacks

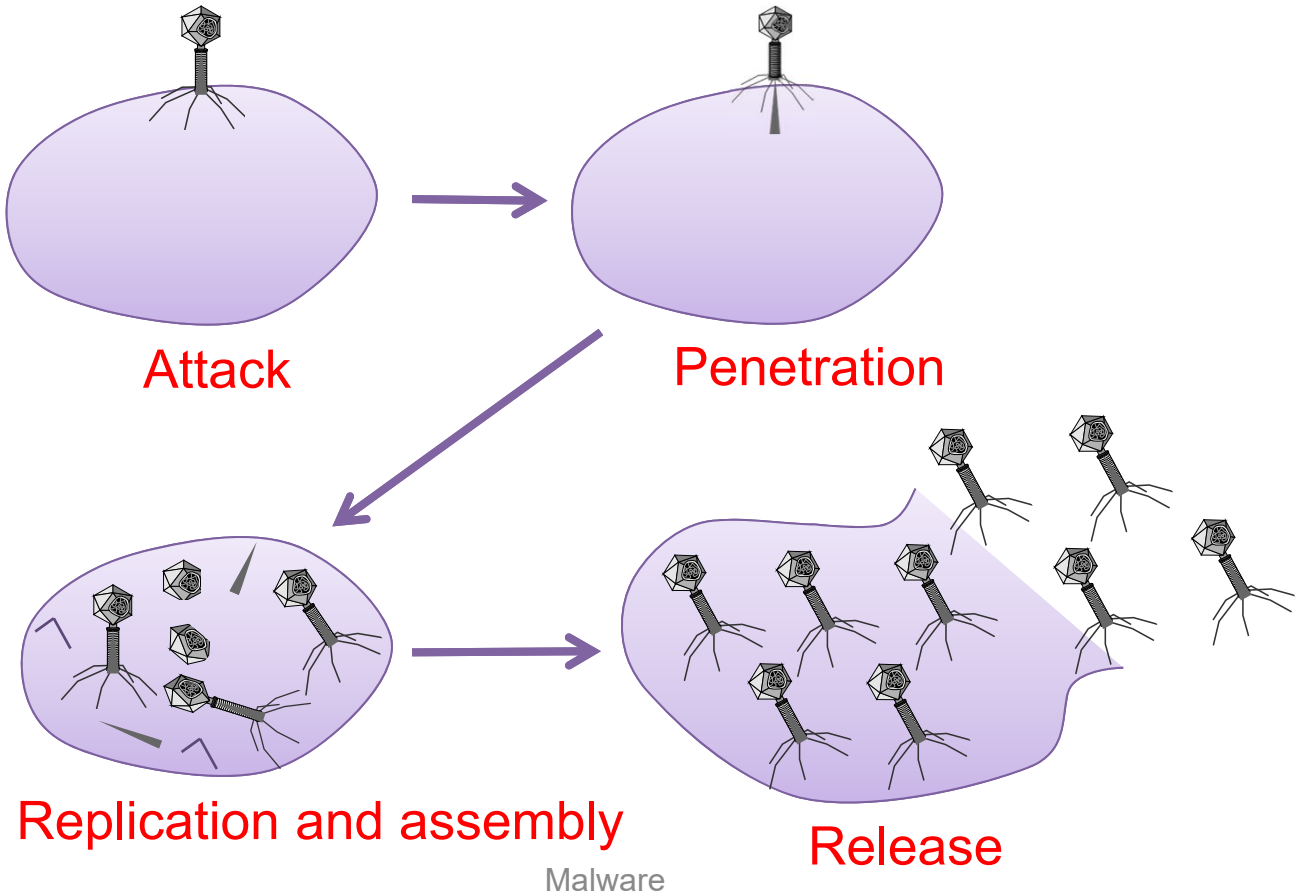
- Avoid single points of failure
- Use code walk-throughs (peer review)
- Use archiving and reporting tools
- Limit authority and permissions (Principle of least Privilege)
- Physically secure critical systems
- Monitor (disgruntled) employee behavior
- Control software installations – reliable sources

Computer Viruses

- A **computer virus** is computer code that can replicate itself by modifying other files or programs to insert code that is capable of further replication.
- This self-replication property is what distinguishes computer viruses from other kinds of malware, such as logic bombs.
- Another distinguishing property of a virus is that replication requires some type of **user assistance,** such as clicking on an email attachment or sharing a USB drive.

Biological Analogy

- Computer viruses share some properties with Biological viruses



Early History

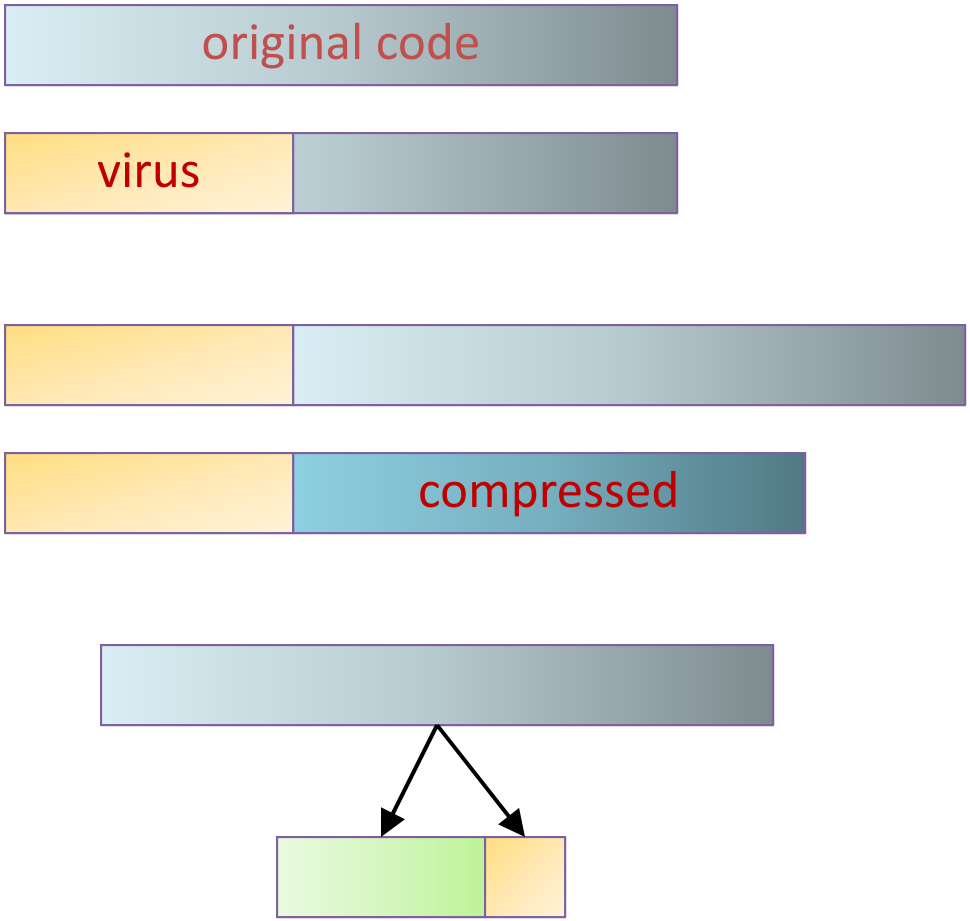
- 1972 sci-fi novel “When HARLIE Was One” features a program called VIRUS that reproduces itself
- First academic use of term virus by PhD student Fred Cohen in 1984, who credits advisor Len Adleman with coining it
- In 1982, high-school student Rich Skrenta wrote first virus released in the wild: Elk Cloner, a boot sector virus
- (c)Brain, by Basit and Amjood Farooq Alvi in 1986, credited with being the first virus to infect PCs

Virus Phases

- **Dormant phase.** During this phase, the virus just exists—the virus is laying low and avoiding detection.
- **Propagation phase.** During this phase, the virus is replicating itself, infecting new files on new systems.
- **Triggering phase.** In this phase, some logical condition causes the virus to move from a dormant or propagation phase to perform its intended action.
- **Action phase.** In this phase, the virus performs the malicious action that it was designed to perform, called **payload.**
  - This action could include something seemingly innocent, like displaying a silly picture on a computer's screen, or something quite malicious, such as deleting all essential files on the hard drive.

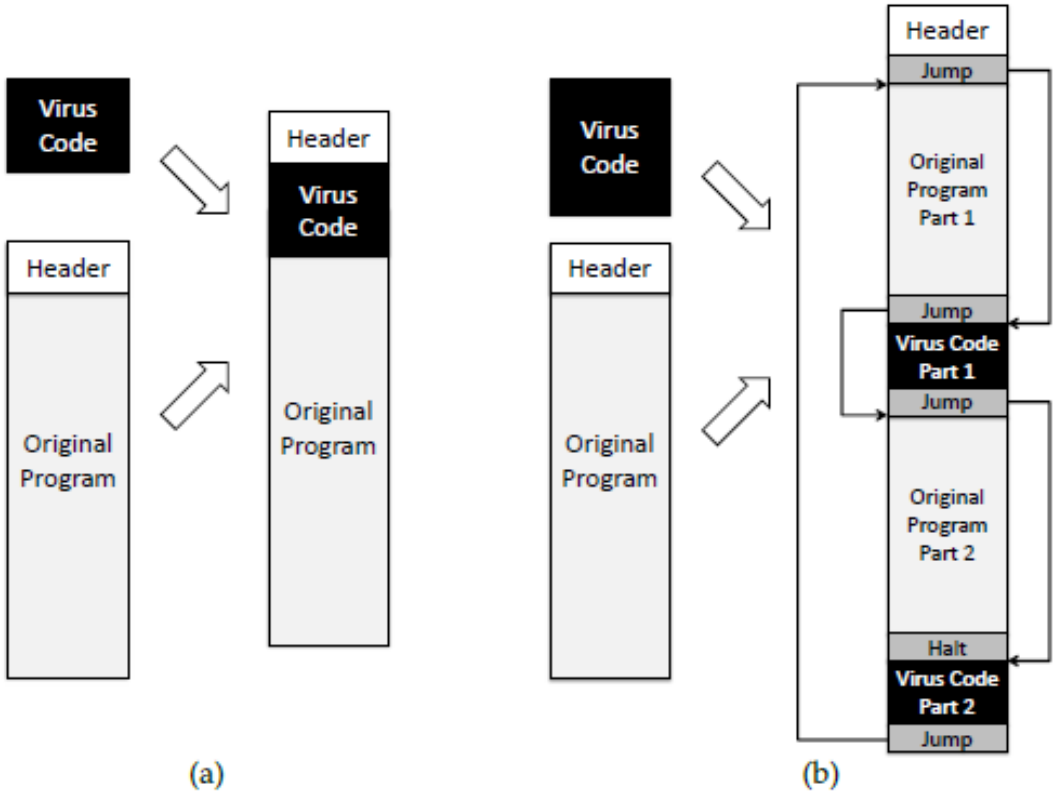
Infection Types

- Overwriting
  - Destroys original code
- Pre-pending
  - Keeps original code, possibly compressed
- Infection of libraries
  - Allows virus to be memory resident
  - E.g., kernel32.dll
- Macro viruses
  - Infects MS Office documents
  - Often installs in main document template



Degrees of Complication

- Viruses have various degrees of complication in how they can insert themselves in computer code.



Case Study: Virus.Dos.Honey.666

- Infect .COM files
- First byte is 0xE9 (meaning jump)
- Malware code is appended at the end of the target
- The first instruction of the target is modified to jump to malware
- The last instruction of malware jumps to the original entry point
- Malware body ends with the byte sequence 0x54, 0x53, which serve as the infection marker
- More detail? https://webpages.uncc.edu/jwei8/Jinpeng_Homepage_files/Virus-DOS-Honey-666-Report.pdf

Offset	Instruction
0000	JMP 0168
…	
0168	PUSH AX
016A	PUSH BP
…	
0280	RET

Offset	Instruction
0000	JMP 0282
…	
0168	PUSH AX
016A	PUSH BP
…	
0280	RET
0282	MOV AX, 10
…	
0630	POP BX
0632	JMP 0168
0636	DB 0x54, 0x53

Computer Worms

- A **computer worm** is a malware program that spreads copies of itself without the need to inject itself in other programs, and usually without human interaction.
- Thus, computer worms are technically not computer viruses (since they don't infect other programs), but some people nevertheless confuse the terms, since both spread by self-replication.
- In most cases, a computer worm will carry a malicious payload, such as deleting files or installing a backdoor.

Early History

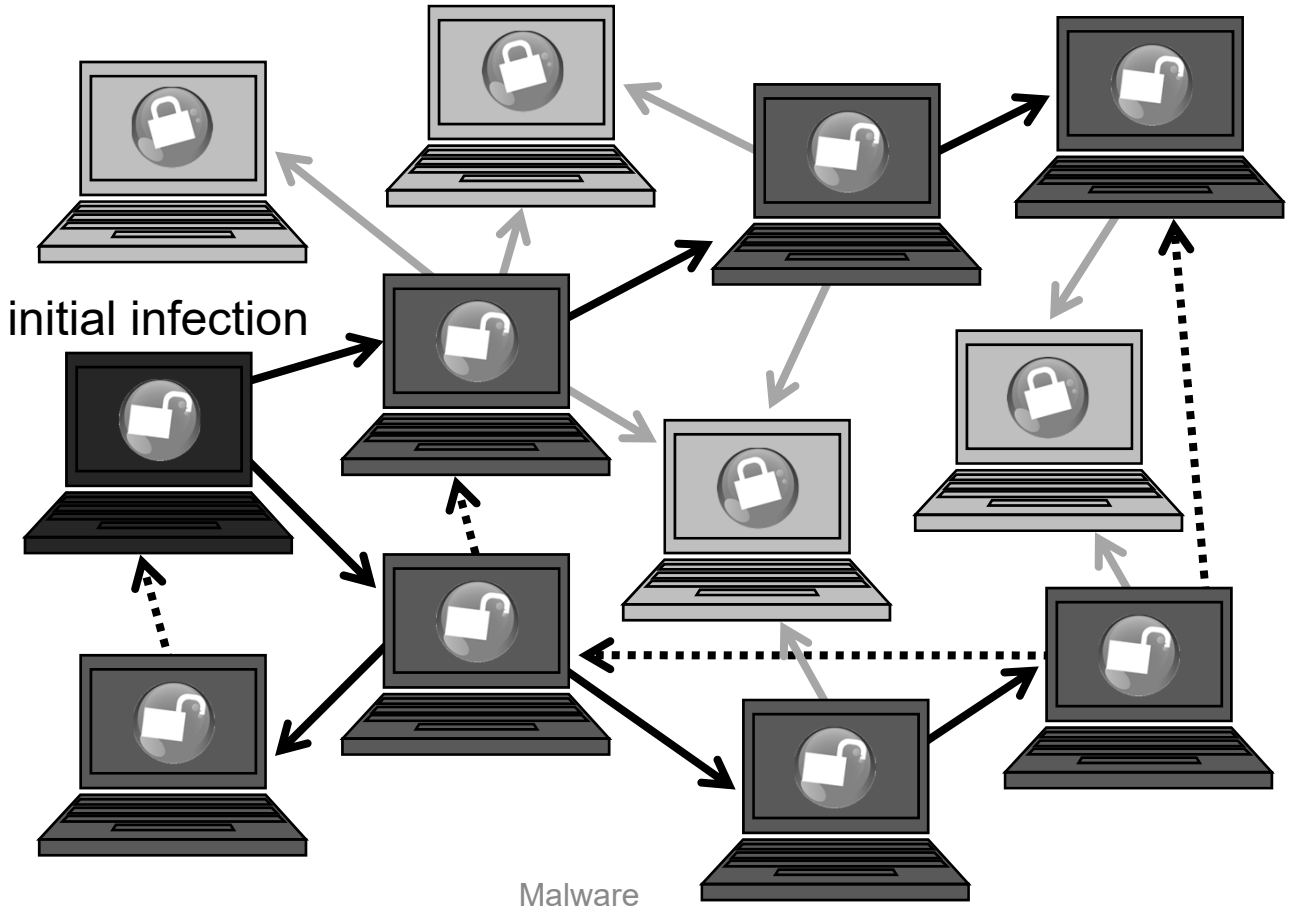
- First worms built in the labs of John Shock and Jon Hepps at Xerox PARC in the early 80s
- CHRISTMA EXEC written in REXX, released in December 1987, and targeting IBM VM/CMS systems was the first worm to use e-mail service
- The first internet worm was the Morris Worm, written by Cornell student Robert Tappan Morris and released on November 2, 1988

Worm Development

- Identify vulnerability still unpatched
- Write code for
  - Exploit of vulnerability
  - Generation of target list
    - Random hosts on the internet
    - Hosts on LAN
    - Divide-and-conquer
  - Installation and execution of payload
  - Querying/reporting if a host is infected
- Initial deployment on botnet
- Worm template
  - Generate target list
  - For each host on target list
    - Check if infected
    - Check if vulnerable
    - Infect
    - Recur
- Distributed graph search algorithm
  - Forward edges: infection
  - Back edges: already infected or not vulnerable

Worm Propagation

- Worms propagate by finding and infecting vulnerable hosts.
  - They need a way to tell if a host is vulnerable
  - They need a way to tell if a host is already infected.



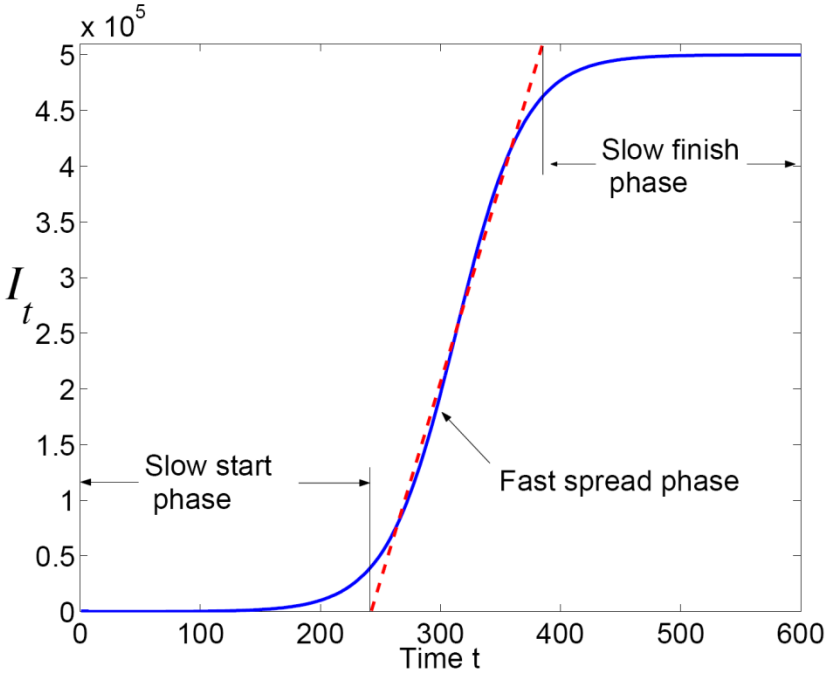
Propagation: Theory

- Classic epidemic model
  - N: total number of vulnerable hosts
  - $I(t)$: number of infected hosts at time t
  - $S(t)$: number of susceptible hosts at time t
  - $I(t) + S(t) = N$
  - β: infection rate
- Differential equation for $I(t)$:

$$dI/dt = \beta I(t)\, S(t)$$

- More accurate models adjust propagation rate over time

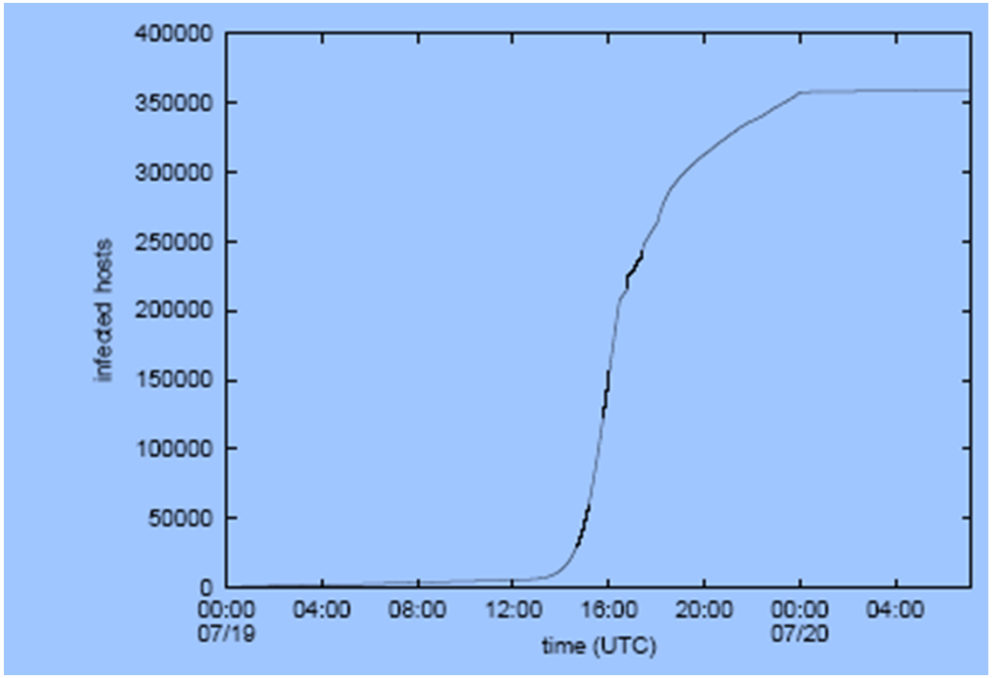
Source:
Cliff C. Zou, Weibo Gong, Don Towsley, and Lixin Gao. The Monitoring and Early Detection of Internet Worms, IEEE/ACM Transactions on Networking, 2005.



Propagation: Practice

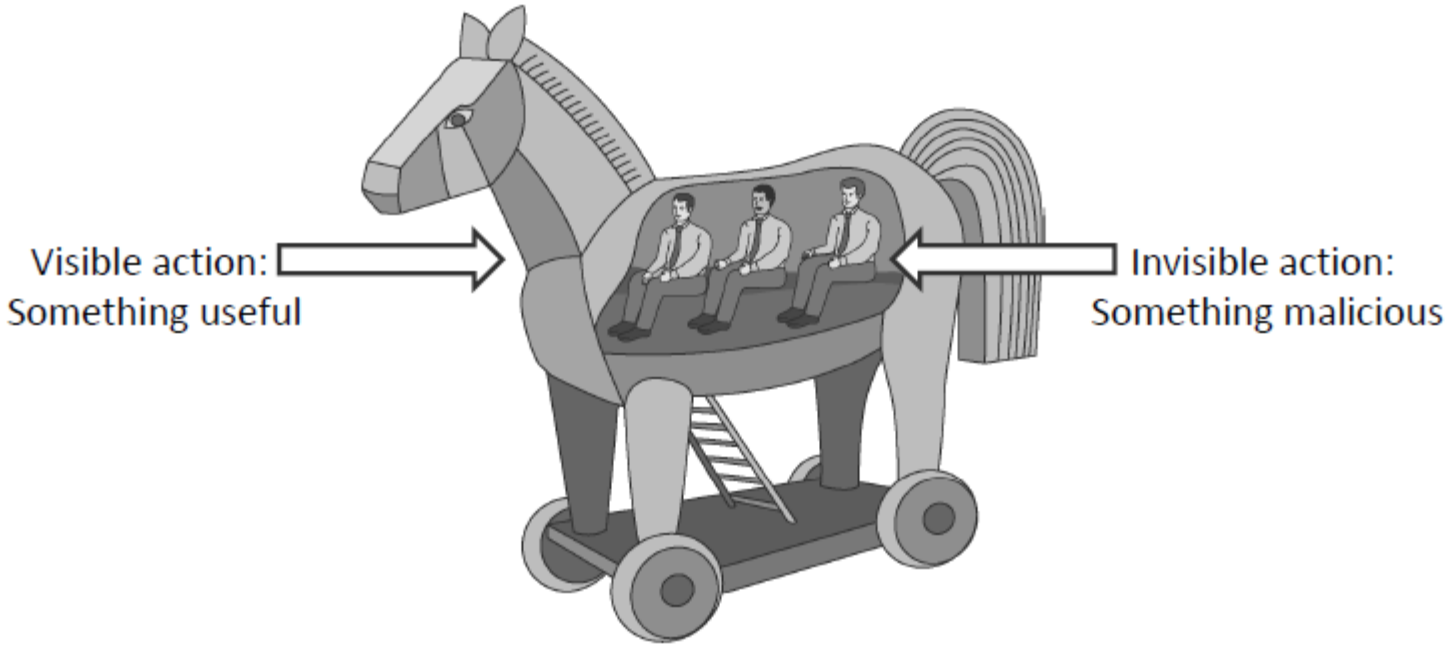
- Cumulative total of unique IP addresses infected by the first outbreak of Code-RedI v2 on July 19-20, 2001

Source:
David Moore, Colleen Shannon, and Jeffery Brown. Code-Red: a case study on the spread and victims of an Internet worm, CAIDA, 2002



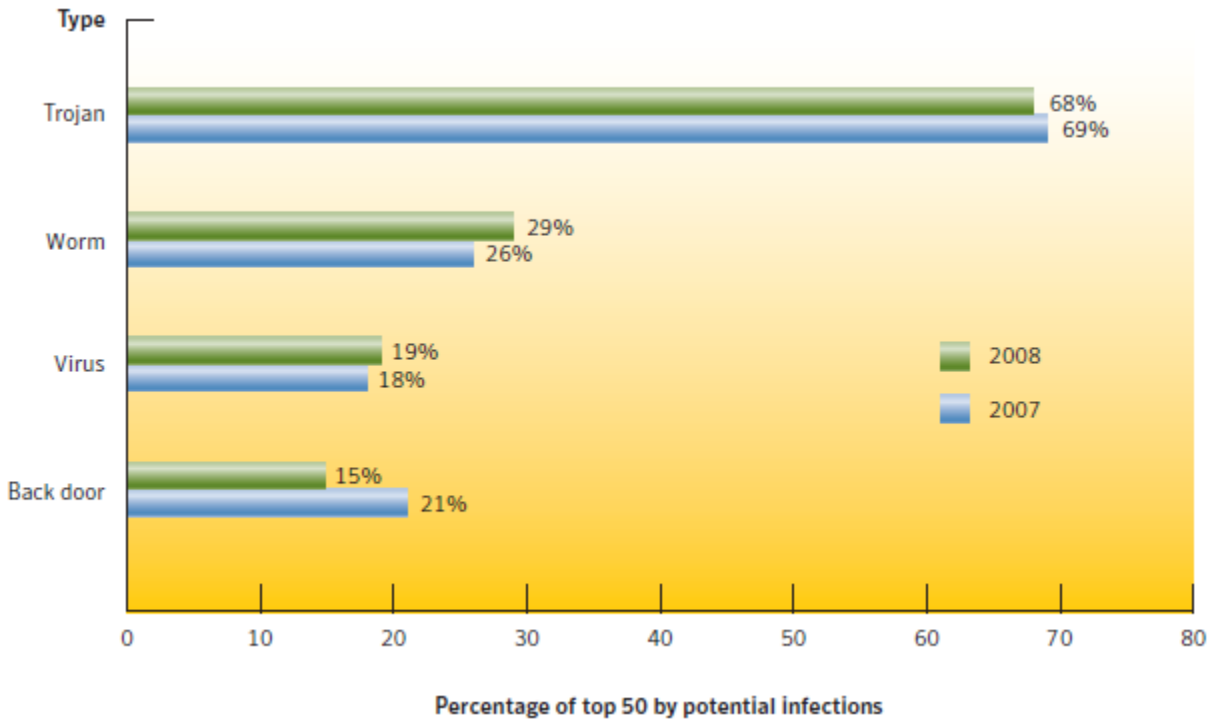
Trojan Horses

- A **Trojan horse (or Trojan)** is a malware program that appears to perform some useful task, but which also does something with negative consequences (e.g., launches a keylogger).
- Trojan horses can be installed as part of the payload of other malware but are often installed by a user or administrator, either deliberately or accidentally.



Current Trends

- Trojans currently have largest infection potential
  - Often exploit browser vulnerabilities
  - Typically used to download other malware in multi-stage attacks



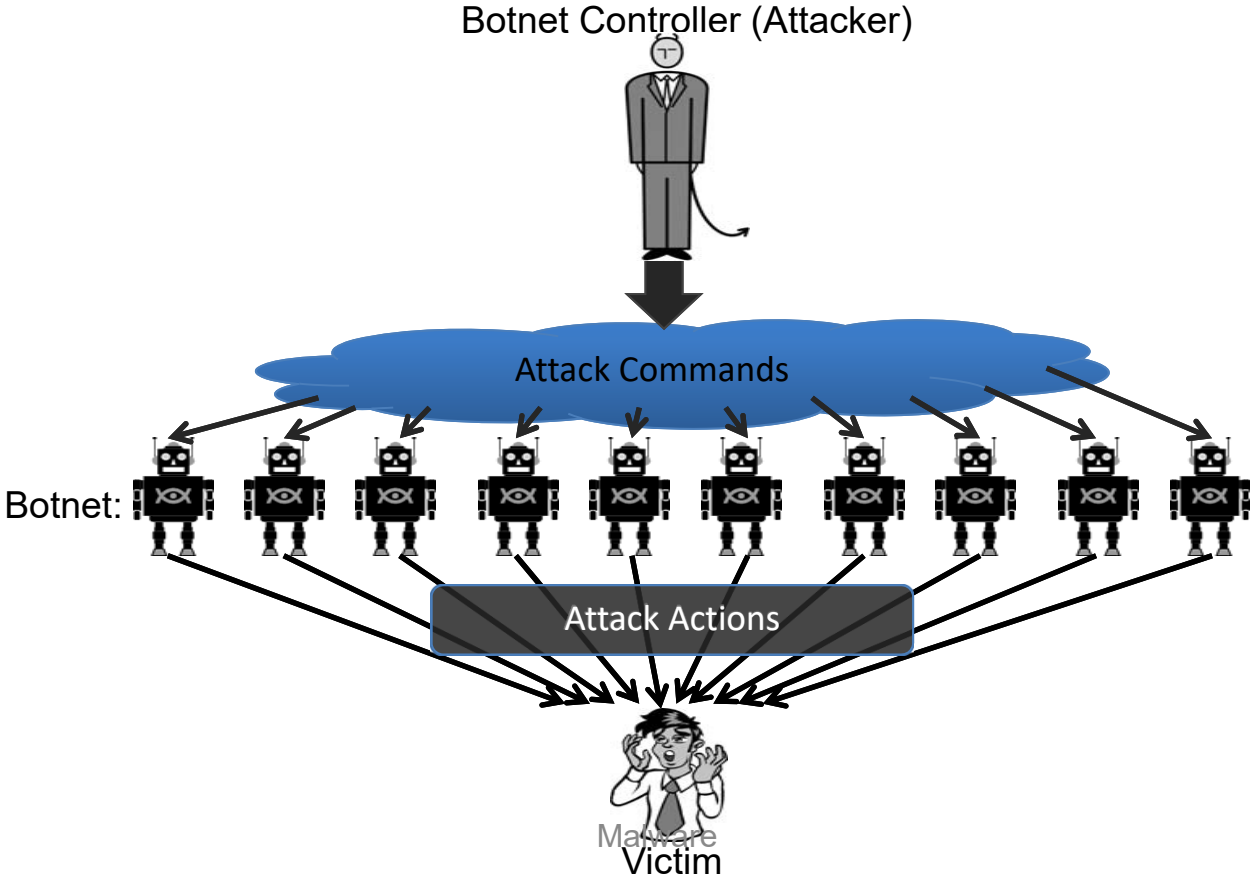
Source:
Symantec Internet Security Threat Report, April 2009

Rootkits

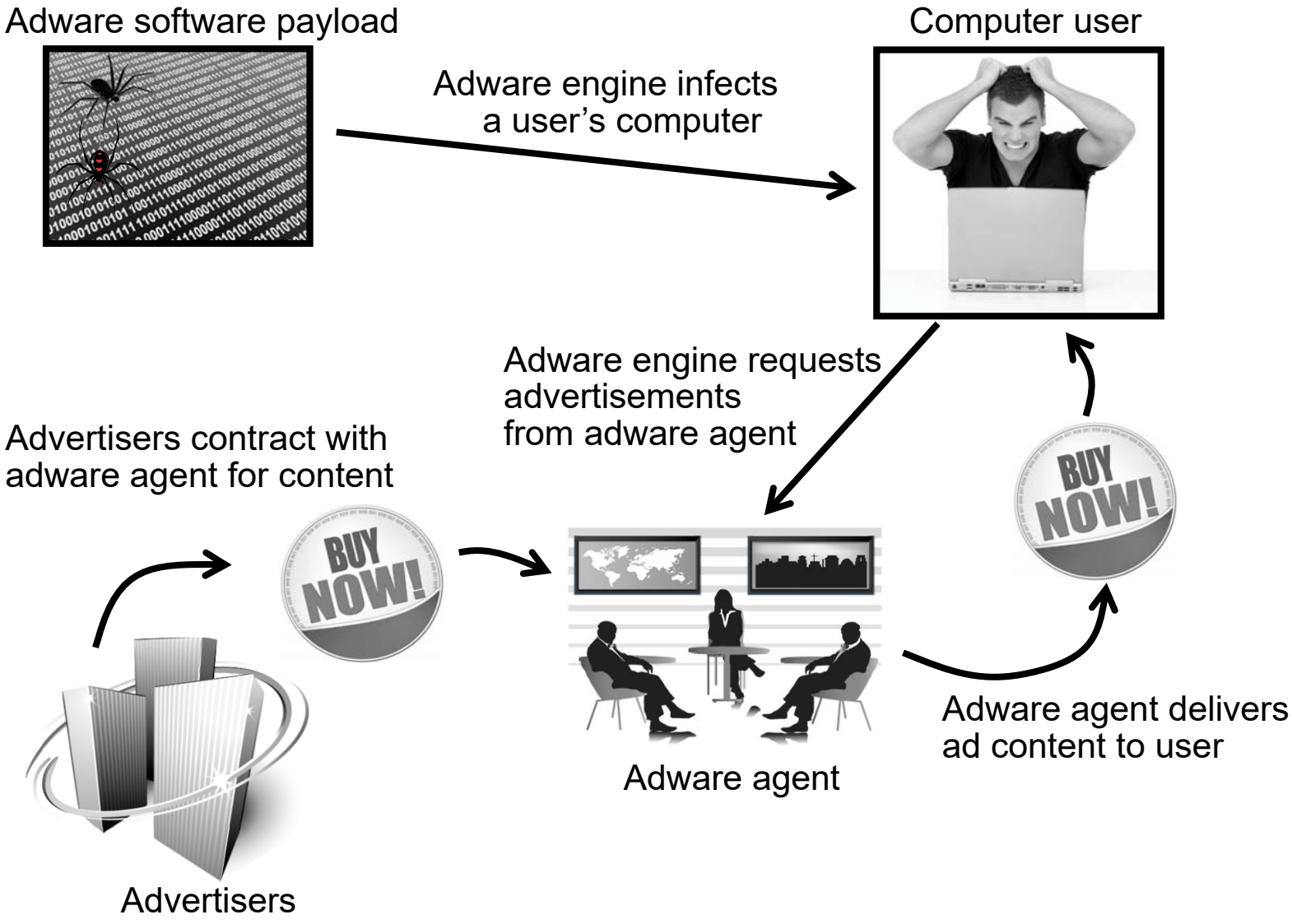
- A rootkit modifies the operating system to hide its existence
  - E.g., modifies file system exploration utilities
  - Hard to detect using software that relies on the OS itself
- RootkitRevealer
  - By Bryce Cogswell and Mark Russinovich (Sysinternals)
  - Two scans of file system
  - High-level scan using the Windows API
  - Raw scan using disk access methods
  - Discrepancy reveals presence of rootkit
  - Could be defeated by rootkit that intercepts and modifies results of raw scan operations

Malware Zombies

- Malware can turn a computer in to a **zombie**, which is a machine that is controlled externally to perform malicious attacks, usually as a part of a **botnet**.

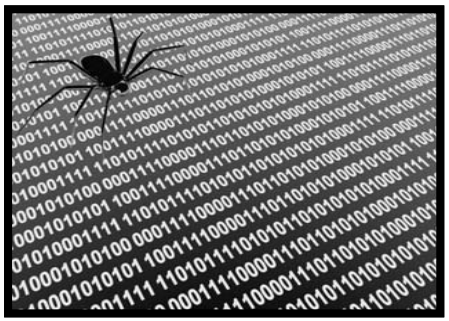


Adware


Adware software payload
Adware engine infects a user's computer
Computer user
Adware engine requests advertisements from adware agent
Advertisers contract with adware agent for content
BUY NOW!
BUY NOW!
Adware agent
Adware agent delivers ad content to user
Advertisers

Spyware

Spyware software payload



1. Spyware engine infects a user's computer.

Computer user

2. Spyware process collects keystrokes, passwords, and screen captures.

3. Spyware process periodically sends collected data to spyware data collection agent.

Spyware data collection agent

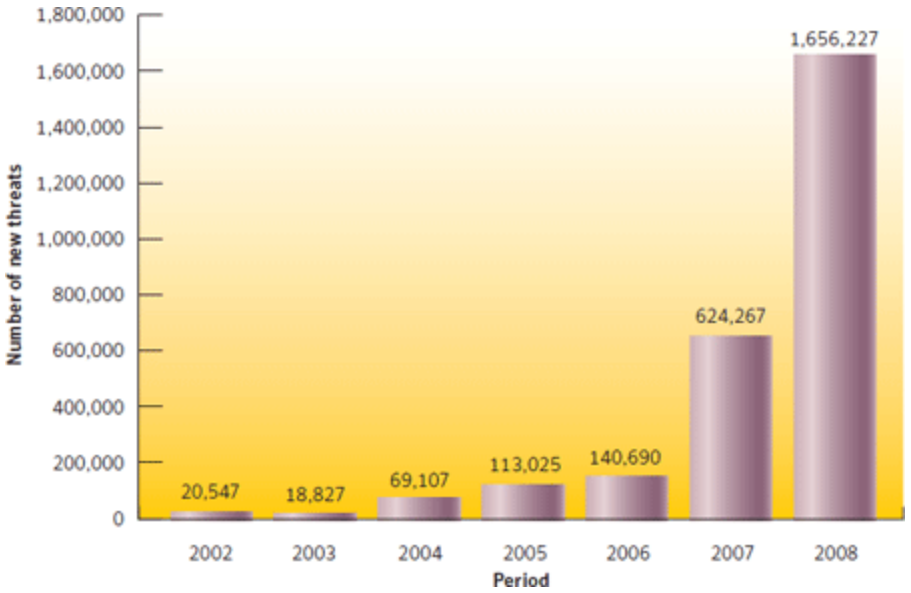
Financial Impact

- Malware often affects a large user population
- Significant financial impact, though estimates vary widely, up to $100B per year (mi2g)
- Examples
  - LoveBug (2000) caused $8.75B in damages and shut down the British parliament
  - In 2004, 8% of emails infected by W32/MyDoom.A at its peak
  - In February 2006, the Russian Stock Exchange was taken down by a virus.

Economics of Malware

- New malware threats have grown from 20K to 1.7M in the period 2002-2008
- Most of the growth has been from 2006 to 2008
- Number of new threats per year appears to be growing an exponential rate.

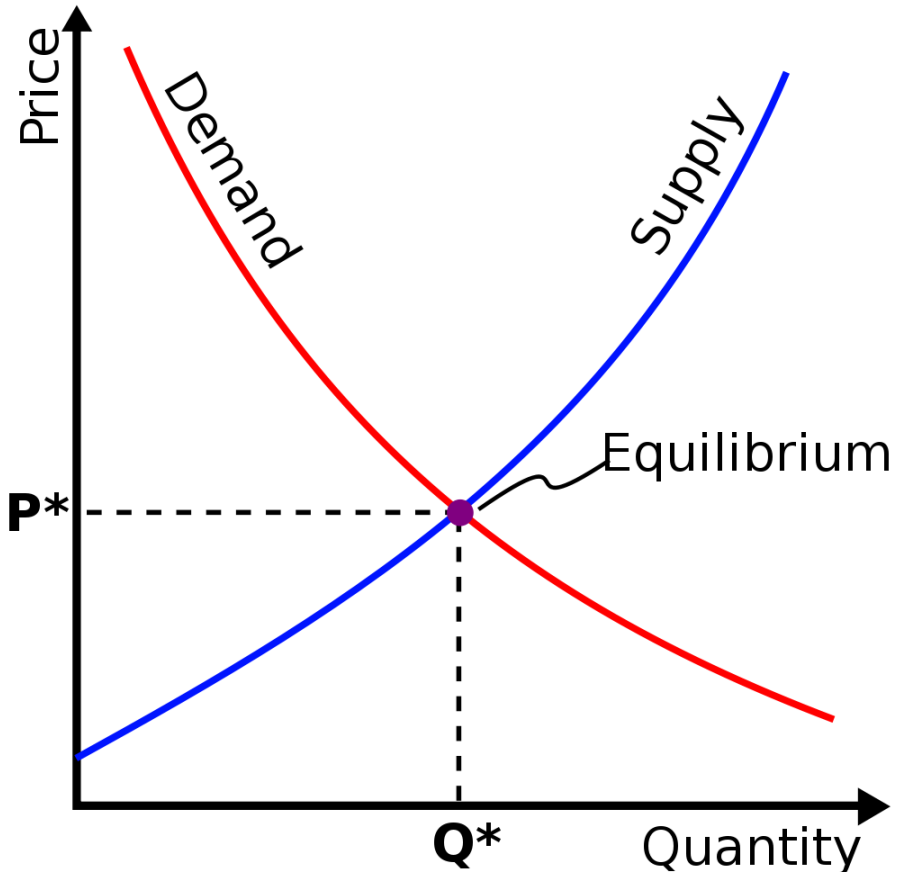
Source:
Symantec Internet Security Threat Report, April 2009



Latest: http://www.symantec.com/threatreport/

Professional Malware

- Growth in professional cybercrime and online fraud has led to demand for professionally developed malware
- New malware is often a custom-designed variations of known exploits, so the malware designer can sell different "products" to his/her customers.
- Like every product, professional malware is subject to the laws of supply and demand.
  - Recent studies put the price of a software keystroke logger at $23 and a botnet use at $225.



Defenses

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

Defenses

- Distinguish between data and instructions
- Limit objects accessible to processes
- 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
  - Key assumptions: (1) certifying authority will *not* make mistakes, and (2) tools and supporting infrastructure used in certifying process are not corrupt

Example: LOCK

- Logical Coprocessor Kernel
  - Designed to meet the highest level of security
- Compiled programs are of 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)

Defenses

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

Limiting Accessibility

- Basis: a user (unknowingly) executes malicious logic, which then executes with all that user's privileges
  - Limiting accessibility of objects should limit spread of malicious logic and effects of its actions
- Approach draws on mechanisms for confinement

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 x is shared, $fd(x)$ increases by 1
  - Whenever $y_1, \ldots, y_n$ used as input to compute z, $fd(z) = \max(fd(y_1), \ldots, fd(y_n))$
- Information x is 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 *the virus* to Bill's program Q
    - Works, as *fd*(P) = 0, so *fd*(Q) = 1 < V_B . Flow distance with respect to the virus
- Cathy executes Q
  - Q tries to write *the virus* to Cathy's program R
    - Fails, as *fd*(Q) = 1, so *fd*(R) would be 2, *fd*(R) ≥ V_C
- 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 intended 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

Multilevel Policies

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

Defenses

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

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

White Listing

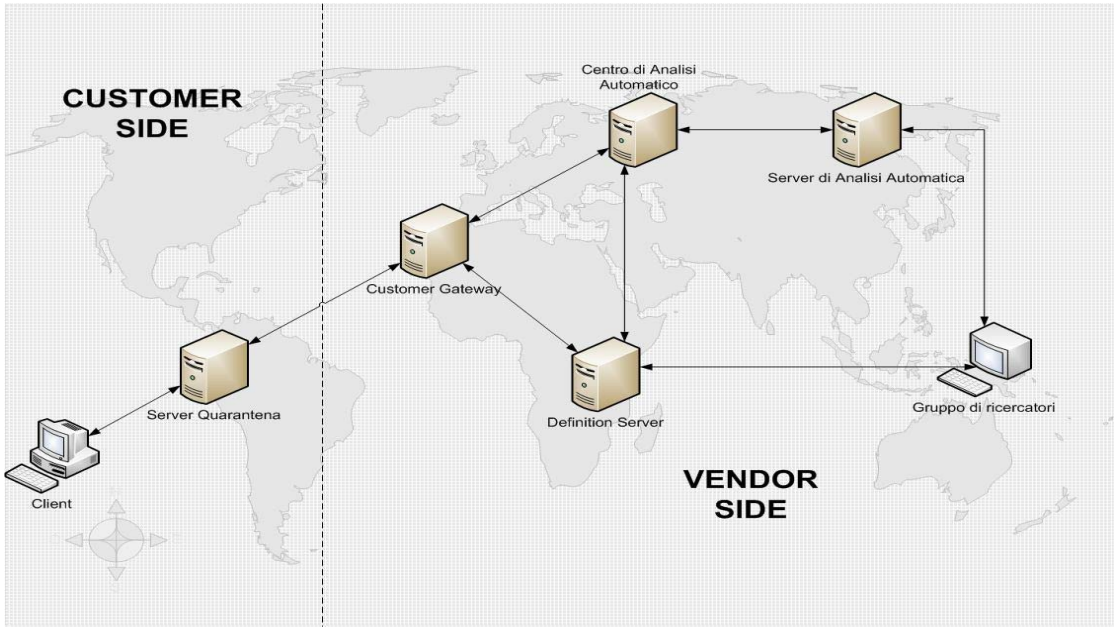
- Maintain database of cryptographic hashes for
  - Operating system files
  - Popular applications
- Compute hash of each file
- Look up into database
- Example: Tripwire, developed in 1992
  - Signature consists of file attributes, cryptographic checksums chosen from among MD4, MD5, HAVAL, SHS, CRC-16, CRC-32, etc.)
- Needs to protect the integrity of the database

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

Signatures Database

- Common Malware Enumeration (CME)
  - aims to provide unique, common identifiers to new virus threats
  - Hosted by MITRE
  - http://cme.mitre.org/data/list.html
- ThreatExpert



MITRE: an American not-for-profit organization

Concealment

- Encrypted malware
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine
- Polymorphic malware
  - Encrypted malware with random variations of the decryption engine (e.g., padding code)
  - Detection using CPU emulator
- Metamorphic malware
  - Different malware bodies
  - Approaches include code permutation and instruction replacement
  - Challenging to detect

Defenses

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

Detect Actions Beyond Spec

- Treat execution, infection as errors and apply fault tolerant techniques

- Example: break program into sequences of nonbranching instructions
  - Checksum each sequence, encrypt result
  - When run, processor recomputes checksum, and at each branch co-processor compares computed checksum with stored one
    - If different, error occurred

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
  - Practical issues: having different algorithms with enough equal intermediate results may be infeasible

Defenses

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

Detecting Statistical Changes

- Example: application had 3 programmers working on it, but statistical analysis (e.g., language, comment styles) shows code from a fourth person—may be from a Trojan horse or virus!

- Other attributes: more conditionals in object code than in source code; 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 (IDS) to detect these

Heuristic Malware Analysis

- Useful to identify new and “zero day” malware
- Code analysis
  - Based on the instructions, the antivirus can determine whether or not the program is malicious, i.e., program contains instruction to delete system files
- Execution emulation
  - Run code in isolated emulation environment
  - Monitor actions that target file takes
  - If the actions are harmful, mark as malware
- Heuristic methods can trigger false alarms

Static vs. Dynamic Analysis

Static Analysis

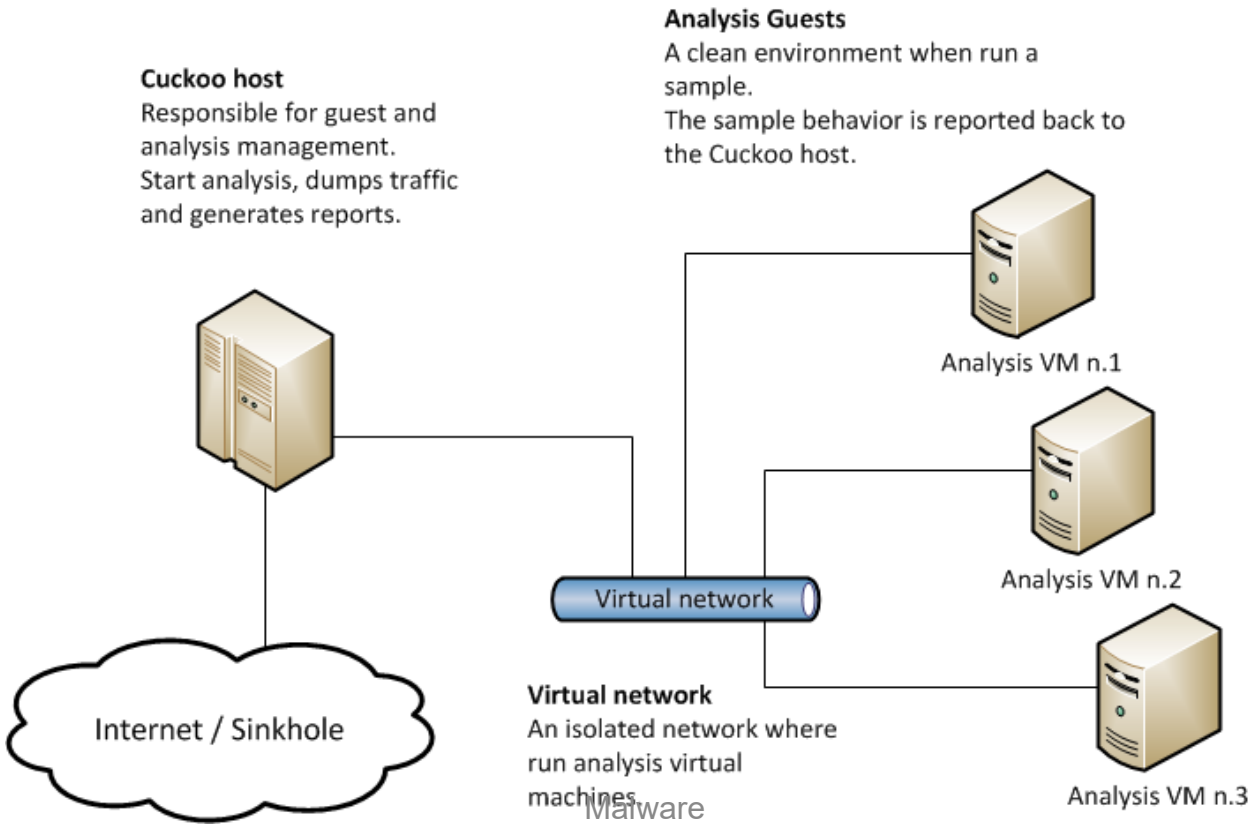
- Checks the code without trying to execute it
- Quick scan in white list
- Filtering: scan with different antivirus and check if they return same result with different name
- Weeding: remove the correct part of files as junk to better identify the virus
- Code analysis: check binary code to understand if it is an executable, e.g., PE
- Disassembling: check if the byte code shows something unusual
- Example: BitBlaze

Dynamic Analysis

- Check the execution of codes inside a virtual sandbox
- Monitor
  - File changes
  - Registry changes
  - Processes and threads
  - Networks ports
- Example systems: Anubis, CWSandbox, Norman Sandbox, ThreatExpert

Dynamic Malware Analysis

- Let malware run and see what it does!
- Safety: in a controlled environment, e.g., a virtual machine



Virus Detection is Undecidable

- Theoretical result by Fred Cohen (1987)
- Virus abstractly modeled as program that eventually executes infect
- Code for infect may be generated at runtime
- Proof by contradiction similar to that of the halting problem

- Suppose program isVirus(P) determines whether program P is a virus
- Define new program Q as follows:

```
if (not isVirus(Q))
  infect;
stop
```

- Running isVirus on Q achieves a contradiction

Other Undecidable Detection Problems

- Detection of a virus
  - by its appearance
  - by its behavior
- Detection of an evolution of a known virus
- Detection of a triggering mechanism
  - by its appearance
  - by its behavior
- Detection of a virus detector
  - by its appearance
  - by its behavior
- Detection of an evolution of
  - a known virus
  - a known triggering mechanism
  - a virus detector

Public Resources on Malware Defense

- ThreatExpert
  - http://www.threatexpert.com/
- Open Malware
  - http://www.offensivecomputing.net/
- Online services
  - Anubis (http://anubis.iseclab.org/)
  - VirusTotal (https://www.virustotal.com/en/)
- Anti-virus companies

More Resources

- Computer Emergency Response Team
  - Research center funded by the US federal government
  - Vulnerabilities database
- Symantec
  - Reports on malware trends
  - Database of malware
- Art of Computer Virus Research and Defense by Peter Szor