Computer and Network Security

R. E. Newman

Computer & Information Sciences & Engineering
University Of Florida
Gainesville, Florida 32611-6120
nemo@cise.ufl.edu

Program Security (Pfleeger Ch. 5)

Software Engineering!

1 Programmed Threats

1.1 Overview

1.1.1 Life forms

1. Bugs
2. Trojan Horses
3. Viruses
4. Worms
5. Bacteria/rabbits

1.1.2 Exposures

1. Trap doors/back doors
2. Information leaks
3. Logic bombs
4. Time bombs

1.2 Viruses

1.2.1 Properties

1. Detection resistant
2. Robust - hard to destroy/deactivate
3. Infectious - wide-ranging, reinfection
4. Easy to create
5. Machine/OS/Application-independent

1.2.2 Dimensions

1. Lifetime
   (a) Transient
   (b) Resident

2. Target
   (a) Boot sector
   (b) TSR code
   (c) Library code
   (d) Application
   (e) Document

3. Attachment method
   (a) Prepended or appended
   (b) Surrounding
   (c) Integrated
   (d) Replaced
      - overwrite
      - change pointers

4. Infection route
   (a) Diskettes/removable media
   (b) Email/MIME
   (c) FTPed/HTTPed files

1.2.3 Virus Signatures

1. Storage patterns
   (a) Modification date
   (b) Size
   (c) Checksum
2. Execution patterns

1.2.4 Virus Defensive Mechanisms

1. Compression
2. Polymorphism
3. Alteration of system utilities

1.2.5 Virus Protection

1. Backups - boot diskette, executables, data files
2. COTS S/W
3. Test new code on isolated machine
4. Virus scanner - update often, use on every diskette coming in *or* going out
5. Access control - limit damage to space accessible by user who ran infected program....
6. H/W-based protection Protected instructions, write protection, base & bounds registgers, VM, tagged memory
7. Audit file signatures

1.2.6 Case Studies

1. Brain

    - Boot sector virus
    - Protected itself in high memory (reset upper memory bound to prevent disturbances)
    - Changed interrupt vectors to screen and redirect disk accesses
    - Marked six disk sectors it used as "faulty" to prevent normal access to them
    - On each disk access, checked if disk uninfected- if so, infect it

2 Worms

2.1 Worm properties

2.2 Xerox Worm

2.3 Internet Worm (1988.11.02)

2.3.1 Modus Operandi

1. Find target hosts
   (a) /etc/hosts file
   (b) .rhost
   (c) hosts.equiv
   (d) at random

2. Gain access
   (a) symmetry of trust (.rhost, hosts.equiv)
   (b) common user accounts/passwords
   (c) password guessing/cracking
   (d) fingerd buffer overflow
   (e) sendmail misconfiguration (DEGUG mode)

3. Launch Grappling Hook (bootstrap loader)
   (a) 99 lines of C code transferred and compiled on target
   (b) given one-time password used to authenticate itself to source machine
   (c) fetch rest of worm from source machine to target - remove traces if any errors
   (d) compile, link, load and execute

4. Hide
   (a) use of one-time password for grappling hook
   (b) encrypt memory-resident copy
   (c) delete all files once in memory
   (d) change name of program periodically
   (e) change PID periodially
   (f) exit before running for too long (note: this make cracking attempts limited per worm)

3 Targeted Malicious Code

3.1 Trapdoors

1. Testing (undocumented ”features”)
2. Maintenance
3. Remote access
4. Intruders
5. Bad code (bounds checking/improper input checking)
6. Undefined machine opcodes

3.2 Salami Attacks

- Collect small amounts of money/time/space so as to be undetected - but many drops of water make up the sea
- Remain because of ... rounding error, poor processes, poor audit

3.3 Covert Channels

3.3.1 Definition

- Information leakage - esp. in multilevel systems
- Legitimately authorized process (e.g., service program run by authorized user) accesses sensitive info
- Program is altered so that it signals unauthorized agent by modulating a medium that this spy can sense, usually in a way that is not readily noticed (noise)

3.3.2 Channel Types

1. Storage channels
   - Presence or absence of objects (e.g., lock on shared object, file, port in use, etc.)
   - Availability of exhaustible resource (e.g., disk space, inodes, etc.)
   - In the noise of accessible data object (e.g., in the last bit of seconds field of time stamps, in the low order bit(s) of images, etc.)

2. Timing channels

- alter usage rates of shared system resources so that another process can tell whether the resource is used heavily or lightly, and interpret these readings as data
- EX - use of time quanta in timeshared system

```
    sender:
use whole quantum = 1
give up CPU immediately = 0
    receiver:
read system clock, interpret bit, then
give up CPU on each time quantum -
big difference between times = 1
small difference between times = 0
```

3.3.3 Covert Channel Identification

1. SRM - Shared Resource Matrix
   (a) Identify all resources that may be read or modified by processes of various classes
   (b) take transitive closure
   (c) look for information flows in violation of policy
   (d) verify flows are real
2. Information flow method
   (a) Determine data and control flows within program
   (b) Determine which outputs are affected by which inputs
   (c) Note: Difficult in face of pointers, recursion

3.3.4 Channel Capacity Estimation

1. Quick & Dirty
   (a) Derive state machine for sending bits (two states, two transitions per state)
   (b) Determine costs of all four transitions
   (c) Take reciprocal of average cost (in time) as channel rate
2. Precise
   Same as above but use information theory result by Shannon as applied by Millen to solve system of linear equations to derive optimal channel rate for symbols with unequal costs

3.3.5 Covert Channel Handling

1. Eliminate flows if possible
   - Partitioned system
2. Reduce flow rates
   - Mode-based systems
   - Dither time
   - Introduce noise
3. Audit
   - Look for attempts to modulate resources

4 Controls for Program Threats

4.1 Development controls - Software Engineering

4.1.1 Peer reviews (walk-throughs)

1. design
2. validation
3. code
4. test plans
5. test results

4.1.2 Cleanroom development

1. program verification
2. front-loaded but faster in practice
3. design for verifiability

4.1.3 Good design practice

1. modularity
2. encapsulation - minimal coupling
3. information hiding
4. code reuse
5. design for testability

4.1.4 Independent testing - separation of duty

4.1.5 Configuration management

1. change control
2. version control
3. backups
4. shadow copies
5. stable versions - code freezes
6. stable configurations - consistency
7. regression testing
8. immutable versions
9. audit trail

4.2 Process Improvement

1. TQM/CQI/CPI/SEI Capability Maturity Models/ISO 9000/1, etc.
    - Structured processes so outcomes are predictable and repeatable

- SEI CMM levels

(a) Initial - chaotic
(b) Repeatable - Planning, islands of process, CM, etc.
(c) Defined - Management support, standardization, documentation, intergroup coordination, peer reviews, training
(d) Managed - Quantitative measures, analysis
(e) Optimizing - feedback

- SSE CMM (System Security Engineering) - NSA extends SEI's CMM 3 areas -
  (a) Engineering (development): SE development, including security analysis vulnerability analysis
  (b) Project (management): quality, assurance
  (c) Organizational: training, process improvement

2. ISO 9000/9001

   - ISO 9000/9001 - general development, installation, maintenance there must exist documented, defined processes (not prescriptive in terms of how good they are)
   - ISO 9000-3 - for software development specifically

3. Issues -

- consistency (precision)
- reliability (accuracy)
- how good a measure are CMM levels of quality, productivity, assurance?

4.3 OS Controls

1. Trusted S/W
   (a) Functional correctness
   (b) enforced integrity
   (c) limited privilege
   (d) appropriate security level

2. Mutual Suspicion
   (a) error checking
   (b) verification of results
   (c) sanity/integrity checks

3. Confinement
   (a) isolation/protection
4. Access log
   (a) audit mechanisms

4.4 Administrative Controls

4.4.1 Standards of program development

1. Design
2. Documentation
3. Programming
4. Coding
5. Testing
6. Configuration management
7. Security audits

4.4.2 Separation of duties

- Chinese Wall Policy
  1. Define Conflict Groups over entities $g : E \to 2^G$ where $G \subseteq 2^E$
  2. Associate objects with entities $e(o)$
  3. Keep history of user accesses $h(u)$, $h : U \to 2^E$
  4. If user u wants to access object o, then allow if

     $$e(o) \in h(u)$$

     else disallow if

     $$e(o) \in \bigcup_{e \in h(u), e' ing(e)} g(e')$$

     else allow and add $e(o)$ to h(u).