Models

- Extreme programming
  - Rapid prototyping and “best practices”
  - Project driven by business decisions
  - Requirements open until project complete
  - Programmers work in teams
  - Components tested, integrated several times a day
  - Objective is to get system into production as quickly as possible, then enhance it
  - Evidence adduced after development needed for assurance
Security: Built In or Add On?

- Think of security as you do performance
  - You don’t build a system, then add in performance later
    - Can “tweak” system to improve performance a little
    - Much more effective to change fundamental algorithms, design
- You need to design it in
  - Otherwise, system lacks fundamental and structural concepts for high assurance
Reference Validation Mechanism

• Reference monitor is access control concept of an abstract machine that mediates all accesses to objects by subjects

• Reference validation mechanism (RVM) is an implementation of the reference monitor concept.
  – Tamperproof
  – Complete (always invoked and can never be bypassed)
  – Simple (small enough to be subject to analysis and testing, the completeness of which can be assured)
    • Last engenders trust by providing assurance of correctness
Examples

- *Security kernel* combines hardware and software to implement reference monitor
- *Trusted computing base (TCB)* is all protection mechanisms within a system responsible for enforcing security policy
  - Includes hardware and software
  - Generalizes notion of security kernel
Adding On Security

- Key to problem: analysis and testing
- Designing in mechanisms allow assurance at all levels
  - Too many features adds complexity, complicates analysis
- Adding in mechanisms makes assurance hard
  - Gap in abstraction from requirements to design may prevent complete requirements testing
  - May be spread throughout system (analysis hard)
  - Assurance may be limited to test results
Example

- 2 AT&T products
  - Add mandatory controls to UNIX system
  - SV/MLS
    - Add MAC to UNIX System V Release 3.2
  - SVR4.1ES
    - Re-architect UNIX system to support MAC
Comparison

• Architecting of System
  – SV/MLS: used existing kernel modular structure; no implementation of least privilege
  – SVR4.1ES: restructured kernel to make it highly modular and incorporated least privilege
Comparison

- File Attributes (*inodes*)
  - SV/MLS added separate table for MAC labels, DAC permissions
    - UNIX inodes have no space for labels; pointer to table added
    - Problem: 2 accesses needed to check permissions
    - Problem: possible inconsistency when permissions changed
    - Corrupted table causes corrupted permissions
  - SVR4.1ES defined new inode structure
    - Included MAC labels
    - Only 1 access needed to check permissions
Key Points

• Assurance critical for determining trustworthiness of systems
• Different levels of assurance, from informal evidence to rigorous mathematical evidence
• Assurance needed at all stages of system life cycle
• Building security in is more effective than adding it later
Malicious Logic Overview

• Defining malicious logic
• Types
  – Trojan horses
  – Computer viruses and worms
  – Other types
• Defenses
  – Properties of malicious logic
  – Trust
Malicious Logic

- 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: NetBus

• Designed for Windows NT 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
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

login source \(\rightarrow\) correct compiler \(\rightarrow\) login executable

user password \(\downarrow\)

logged in

user password or
magic password

login source \(\rightarrow\) doctored compiler \(\rightarrow\) login executable

logged in
The Compiler

compiler source → correct compiler → compiler executable

login source

correct login executable

compiler source → doctored compiler → compiler executable

login source

rigged login executable
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;
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) 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

- **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
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

```
Virus code → Deciphering routine → Enciphered virus code
Deciphering key
```
Example

(* Decryption code of the 1260 virus *)
(* initialize the registers with the keys *)
\[ rA = k1; \quad rB = k2; \]

(* initialize \( rC \) with the virus;
    starts at sov, ends at eov *)
\[ rC = sov; \]

(* the encipherment loop *)
while \((rC \neq eov)\) do begin
  (* encipher the byte of the message *)
  \[ (*rC) = (*rC) \text{ xor } rA \text{ xor } rB; \]
  (* 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)
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