Malicious Logic

- Chapter 19
- What is malicious logic
- Types of malicious logic
- Defenses

History

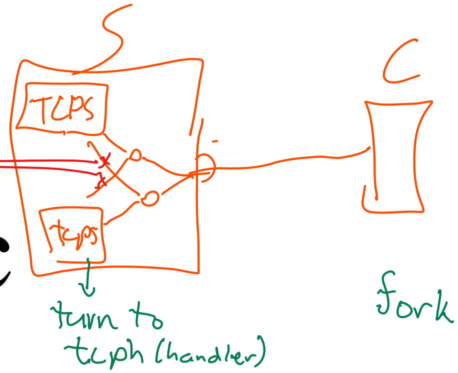
- Password cracker
  - Since we have secrets: word puzzle
  - Attack techniques : crypto-analysis, brute force
- Virus
  - Since we have computers
    - virus, pop-ups
    - elicit users to execute a copied or downloaded program
  - Attack techniques : replace system files, hook to system processes, memory leak

History

- Worm
  - Since we have the Internet
    - Our computers do nothing but keep online and get infected.
  - Attack techniques : buffer overflow
- Recent
  - Rootkits: kernel and library modification
  - Web attacks: SQL injection, cross-site scripting
  - VM attacks: hypervisor exploitation

want to remove it. still listening

Malicious Logic



- Set of instructions that cause site security policy to be violated
- Example
  - Shell script “ls” on a UNIX system:

```
cp /bin/sh /tmp/.xyzzy
chmod u+s,o+x /tmp/.xyzzy
rm ./ls
ls $*
```

Example

- Attack procedure
  - Place in program called “ls”
  - Make “.” in a user's PATH environment
  - Wait for or trick the user to execute “ls”
  - Execute “`/tmp/.xyzzy`”
  - You now have a shell with the permission of the user.

Malicious Logic

- Security policy states the secure states.
  - All programs shall run as the login user.
- Implemented mechanisms make the restricted states.
  - . is added into PATH.
  - A program can be made as the non-login user.
- Malicious logic finds the existence of the restricted but not secure states and the approach to get into such states.

Outline

- What is malicious logic
- Types of malicious logic
  - Trojan House
  - Computer Virus
  - Computer Worms
  - Denial of Service (DoS)
  - Steganography
  - Web exploit
- Defenses

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
  - Case 18 to demo

To Survive

- Trojan programs do not survive when they are replaced by newer versions (update)
- To survive
  - Trojan programs make copies of itself to other programs
    - Also called propagating Trojan horse
  - Make the update program as a Trojan

Update Trojan

- Trojan update program
  - Yum/dnf, apt-get, windows update
  - Overt: update software
  - Covert: insert a Trojan into the updated program to make them Trojans
  - Problem: cannot survive when the update program updates itself
    - Another covert: insert a Trojan into the updated update program

Outline

- What is malicious logic
- Types of malicious logic
  - Trojan House
  - Computer Virus
  - Web exploit
  - Computer Worms
  - Denial of Service (DoS)
  - Steganography
- Defenses

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:
```

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

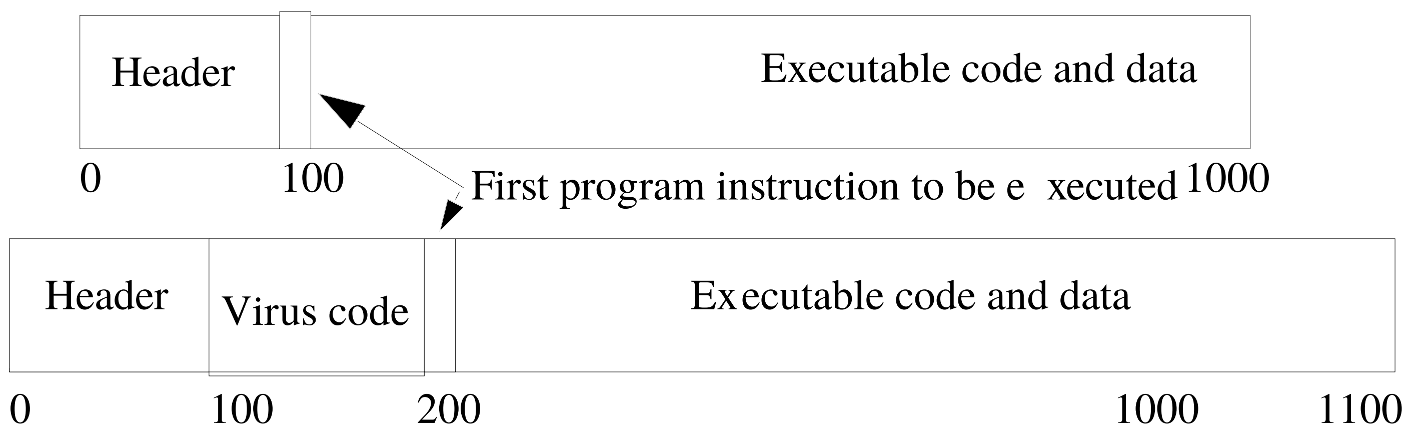
Types of Viruses

- Boot sector infectors
- Executable infectors
- TSR viruses
- Stealth viruses
- Encrypted viruses
- Polymorphic viruses
- Macro viruses
- Logic bomb

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.

TSR Viruses

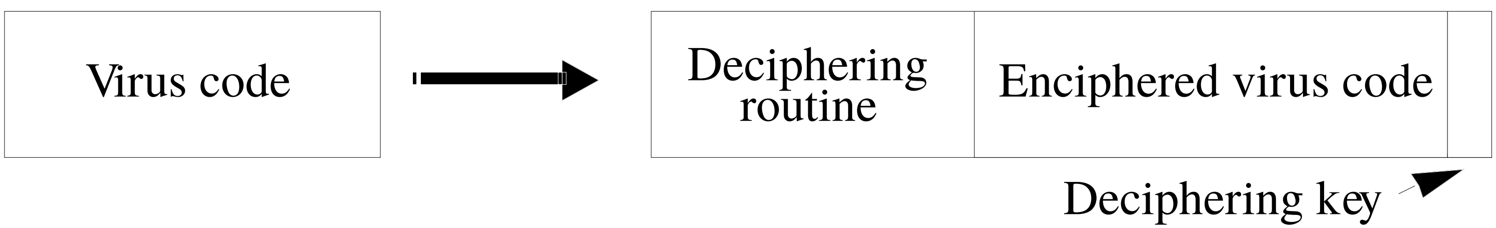
- A virus that stays active in memory after the application (or bootstrapping, or disk mounting) is completed
  - TSR is "Terminate and Stay Resident"
  - Case 18, trojan
- Examples: Brain, Jerusalem viruses
  - Stay in memory after program or disk mount is completed

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



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)

Example

- These are different instructions (with different bit patterns) but have the same effect:
  - add, sutract, xor 0 to register
  - no-op
  - k=3 : k=1 ; k++; k*=2; k--;
  - k=3 : j=2 ; k=2*j-1;
- Polymorphic virus would pick randomly from among these instructions
- Code obfuscation (case10)

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

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

Outline

- What is malicious logic
- Types of malicious logic
  - Trojan House
  - Computer Virus
  - Web exploit
  - Computer Worms
  - Denial of Service (Dos)
  - Steganography
- Defenses

Web Exploit

- Web architecture
  - Client side
    - HTML: form (data), DOM (html)
    - CSS: display of html
    - Javascript: build dynamic html on client side
  - Server side
    - Httpd service: broker
    - Web app: php, python, perl, ruby, ...
    - Database: mysql, postgres, mogodb, ...

Web Exploit

- Web app exploit
  - Form data injection
  - Cookie/session alteration/hijacking
- Database exploit
  - SQL injection
  - Case14
- Client side exploit
  - Javascript injection (through XXS)

Web Coding Flaws

- SQL statement with string concatenation
  - In web app (PHP, Perl, Java, Python, Ruby ...)
  - "select * from users where id=" + id + ";"
- Suggested solutions
  - Object-relation mapping (ORM)
    - No direct SQL coding
  - Parameterized SQL statement
  - Input sanity check

Web Coding Flaws

- Dynamic HTML with html() or string concatenation or embedded url or ...
  - In client (Javascript)
  - $('td#userid').html(id)
  - $('tr#user').append($('<td>'+id+</td>'))
- Typical solutions:
  - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Outline

- What is malicious logic
- Types of malicious logic
  - Trojan House
  - Computer Virus
  - Web exploit
  - Computer Worms
  - Denial of Service (Dos)
  - Steganography
- Defenses

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: Morris Worm, 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

Example: Christmas Worm

- Distributed in 1987, designed for IBM networks
- Electronic letter instructing recipient to save it and run it as a program
  - Drew Christmas tree, printed "Merry Christmas!"
  - Also checked address book, list of previously received email and sent copies to each address
- Shut down several IBM networks
- Really, a macro worm
  - Written in a command language that was interpreted

Outline

- What is malicious logic
- Types of malicious logic
  - Trojan House
  - Computer Virus
  - Web exploit
  - Computer Worms
  - Denial of Service (Dos)
  - Steganography
- Defenses

DoS

- 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

Resource List

- CPU cycles
- Memory
- Network bandwidth
- Hard drive
- Limited data pool
  - TCP port numbers
  - DHCP pool
  - I-node table

Outline

- What is malicious logic
- Types of malicious logic
  - Trojan House
  - Computer Virus
  - Computer Worms
  - Denial of Service (Dos)
  - Steganography
  - Web exploit
- Defenses

Steganography

- Hide data in other data
  - The hidden data is not encrypted
  - The hidden is mixed within other data
- Carrier: file, message, multimedia
- Typical approaches
  - Append to a carrier
  - Embed as an object in a carrier
  - Trivial modification over data in a carrier
- Case21

Outline

- What is malicious logic
- Types of malicious logic
- Defenses

Difficulties

- Not just anti-virus or patch
- Unknown flaws
- Unknown malicious logic
- Indistinguishable data and code
- Unpredictable subjects and objects
- Changing policy

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)

Example: Exec-Shield

- Stack overflow
- Original security policy
  - A program can fully access (rwx) its memory.
  - Code is stored as data in stack.
- New security policy and mechanism
  - Stack is made not executable
  - Code in stack is thus not executable

Data vs. Instructions

- Mutual exclusion of write and execute
- ACM
  - Objects: data objects, instruction objects
  - Subjects:
  - Rights: rw, rx
    - no wx
  - Transition: change the type of objects

Defenses

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

Limiting Accessibility

- Limiting accessibility of objects should limit spread of malicious logic and effects of its actions
- Approaches on confinement
  - Information flow
  - Reduce protection domain (least privilege)
  - Sandboxing
  - Randomization

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 $y_1, \ldots, y_n$ used as input to compute z, $fd(z) = \max(fd(y_1), \ldots, fd(y_n))$
- 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

Information Flow Metrics

- Rights are given based on the IF metrics
- ACM
  - Objects: objects with different IF metrics
  - Subjects
  - Rights: determined by IF metrics
  - Transition: change of IF metrics

Reducing Protection Domain

- Application of principle of least privilege
- Basic idea: remove rights from process so it can only perform its function
  - If a function requires write, it may write to anything
  - But you can make sure it writes only to those objects you expect

Reducing Protection Domain

- Subjects are assigned with needed rights
- ACM
  - Objects:
  - Subjects: dynamically created
  - Rights: assigned according to needs
  - Transition: no inheritage in newly created subjects

Sandboxing

- Sandboxes, virtual machines also restrict rights
  - Modify program by inserting instructions to cause traps when violation of policy
  - Replace dynamic load libraries with instrumented routines

Example: Randomization

- Original policy (no such policy in fact)
  – A process can execute code in stack
- New policy
  – A process shall not execute code in stack
- Mechanism
  – Make it hard for a process to execute code in stack

Example: Randomization

- ACM
  - Objects: memory pages
  - Subjects:
  - Rights: assigned randomly to objects
  - Transition:

Defenses

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

Inhibit Sharing

- Separation of duty/domain to ensure integrity
- Do not directly share data and procedures in one domain to other domains.
- Rather, to share, the data/procedure is copied to a protected domain dedicated for sharing.
- If the copied and shared data/procedure in the protected domain is compromised, it will not affect the original domain.
- Example: LOCK keeps single copy of shared procedure in memory.

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
- Example: tripwire
  - Signature consists of file attributes, cryptographic checksums chosen from among MD4, MD5, HAVAL, SHS, CRC-16, CRC-32, etc.)
  - National Software Reference Library

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

Defenses

- Distinguish between data, instructions
- Limit objects accessible to processes
- Inhibit sharing
- 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
  - Different algorithms with enough equal intermediate results may be infeasible
    - Especially for malicious logic, where you would check file accesses

Proof-Carrying Code

- Example : Stack Guard
- Put a canary before a return address.
- Stack overflow must overwrite the canary in order to overwrite the return address.
- Canary
  – Termination canary
    - 0x000D0AFF : NULL, CR, LF, -1
  – Random canary
    - 0xXXXXXXXX

Defenses

- Distinguish between data, instructions
- Limit objects accessible to processes
- Inhibit sharing
- 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 shows code from a fourth person—may be from a Trojan horse or virus!
- Other attributes: more conditionals than in original; 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 to detect these