Lecture 22
November 20, 2023

Defenses

- Scanning
- Distinguishing between data, instructions
- Containing
- Specifying behavior
- Limiting sharing
- Statistical analysis

Scanning Defenses

- Malware alters memory contents or disk files
- Compute manipulation detection code (MDC) to generate signature block for data, and save it
- Later, recompute MDC and compare to stored MDC
  - If different, data has changed

Example: *tripwire*

- File system scanner
- Initialization: it computes signature block for each file, saves it
  - Signature consists of file attributes, cryptographic checksums
  - System administrator selects what file attributes go into signature
- Checking file system: run *tripwire*
  - Regenerates file signatures
  - Compares them to stored file signatures and reports any differences

Assumptions

- Files do not contain malicious logic when original signature block generated
- Pozzo & Grey: implement Biba's model on LOCUS to make assumption explicit
  - Credibility ratings assign trustworthiness numbers from 0 (untrusted) to *n* (signed, fully trusted)
  - Subjects have risk levels
    - Subjects can execute programs with credibility ratings ≥ risk level
    - If credibility rating < risk level, must use special command to run program

Antivirus Programs

- Look for specific “malware signatures”
  - If found, warn user and/or disinfect data
- At first, static sequences of bits, or patterns; now also includes patterns of behavior
- At first, derived manually; now usually done automatically
  - Manual derivation impractical due to number of malwares

Example: Earlybird

- System for generating worm signatures based on worm increasing network traffic between hosts, and this traffic has many common substrings
- When a packet arrives, its contents hashed and destination port and protocol identifier appended; then check hash table (called *dispersion table*) to see if this content, port, and protocol have been seen
    - If yes, increment counters for source, destination addresses; if both exceed a threshold, content may be worm signature
    - If no, run through a multistage filter that applies 4 different hashes and checks for those hashes in different tables; count of entry with smallest count incremented; if all 4 counters exceed a second threshold, make entry in dispersion table
- Found several worms before antimalware vendors distributed signatures for them

Example: Polygraph

- Assumes worm is polymorphic or metamorphic
- Generates classes of signatures, all based on substrings called *tokens*
  - *Conjunction signature*: collection of tokens, matched if all tokens appear regardless of order
  - *Token-subsequence signature*: like conjunction signature but tokens must appear in order
- Bayes signature associates a score with each token, and threshold with signature
  - If probability of the payload as computed from token scores exceeds a threshold, match occurs
- Experimentally, Bayes signatures work well when there is little non-malicious traffic, but if that's more than 80% of network traffic, no worms identified

Behavioral Analysis

- Run suspected malware in a confined area, typically a sandbox, that simulates environment it will execute in
- Monitor it for some time period
- Look for anything considered "bad"; if it occurs, flag this as malware

Example: Panorama

- Loads suspected malware into a Windows system, which is itself loaded into Panorama and run
  - Files belonging to suspect program are marked
- Test engine sends “sensitive” information to trusted application on Windows
- Taint engine monitors flow of information around system
  - So when suspect program and trusted application run, behavior of information can be recorded in taint graphs
- Malware detection engine analyzes taint graphs for suspicious behavior
- Experimentally, Panorama tested against 42 malware samples, 56 benign samples; no false negatives, 3 false positives

Evasion

Malware can try to ensure malicious activity not triggered in analysis environment

- Wait for a (relatively) long time
- Wait for a particular input or external event
- Identify malware is running in constrained environment
  - Check various descriptor tables
  - Run sequence of instructions that generate an exception if not in a virtual machine (in 2010, estimates found 2.13% of malware samples did this)

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 are assumption that certifying authority will *not* make mistakes and assumption that tools, supporting infrastructure used in certifying process are not corrupt

Example: Duff and UNIX

- Observation: users with execute permission usually have read permission, too
  - So files with “execute” permission have type “executable”; those without it, type “data”
  - Executable files can be altered, but type immediately changed to “data”
    - Implemented by turning off execute permission
    - Certifier can change them back
      - So virus can spread only if run as certifier

Containment

- 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 information x has $fd(x) = 0$
  - Whenever information 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

Implementation Issues

- Metric associated with *information*, not *objects*
  - You can tag files with metric, but how do you tag the information in them?
  - This inhibits sharing
- To stop spread, make V = 0
  - Disallows sharing
  - Also defeats purpose of multi-user systems, and is crippling in scientific and developmental environments
    - Sharing is critical here

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

Example: ACLs and C-Lists

- s_1 owns file f_1 and s_2 owns program p_2 and file f_3
  - Suppose s_1 can read, write f_1, execute p_2, write f_3
  - Suppose s_2 can read, write, execute p_2 and read f_3
- s_1 needs to run p_2
  - p_2 contains Trojan horse
    - So s_1 needs to ensure p_{12} (subject created when s_1 runs p_2) can't write to f_3
  - Ideally, p_{12} has capability { (s_1, p_2, x) } so no problem
    - In practice, p_{12} inherits s_1's rights, so it can write to f_3—bad! Note s_1 does not own f_3, so can't change its rights over f_3
- Solution: restrict access by others

Authorization Denial Subset

- Defined for each user s_i
- Contains ACL entries that others cannot exercise over objects s_i owns
- In example: $R(s_2) = \{ (s_1, f_3, w) \}$
  - So when p_{12} tries to write to f_3, as p_{12} owned by s_1 and f_3 owned by s_2, system denies access
- Problem: how do you decide what should be in your authorization denial subset?

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

Lai and Gray

- Implemented modified version of Karger's scheme on UNIX system
  - Allow programs to access (read or write) files named on command line
  - Prevent access to other files
- Two types of processes
  - Trusted: no access checks or restrictions
  - Untrusted: valid access list (VAL) controls access and is initialized to command line arguments plus any temporary files that the process creates

File Access Requests

1. If file on VAL, use effective UID/GID of process to determine if access allowed
2. If access requested is read and file is world-readable, allow access
3. If process creating file, effective UID/GID controls allowing creation
    - Enter file into VAL as NNA (new non-argument); set permissions so no other process can read file
4. Ask user. If yes, effective UID/GID controls allowing access; if no, deny access

Example

- Assembler invoked from compiler
- as x.s /tmp/ctm2345
- and creates temp file /tmp/as1111
  - VAL is
  - x.s /tmp/ctm2345 /tmp/as1111
- Now Trojan horse tries to copy x.s to another file
  - On creation, file inaccessible to all except creating user so attacker cannot read it (rule 3)
  - If file created already and assembler tries to write to it, user is asked (rule 4), thereby revealing Trojan horse

Trusted Programs

- No VALs applied here
  - UNIX command interpreters: *csh*, *sh*
  - Program that spawn them: *getty*, *login*
  - Programs that access file system recursively: *ar*, *chgrp*, *chown*, *diff*, *du*, *dump*, *find*, *ls*, *restore*, *tar*
  - Programs that often access files not in argument list: *binmail*, *cpp*, *dbx*, *mail*, *make*, *script*, *vi*
  - Various network daemons: *fingerd*, *ftpd*, *sendmail*, *talkd*, *telnetd*, *tftpd*

Specifications

- Treat infection, execution phases of malware as errors
- Example
  - Break programs into sequences of non-branching instructions
  - Checksum each sequence, encrypt it, store it
  - When program is run, processor recomputes checksums, and at each branch compares with precomputed value; if they differ, an error has 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

Inhibit Sharing

- Use separation implicit in integrity policies
- Example: LOCK keeps single copy of shared procedure in memory
  - Master directory associates unique owner with each procedure, and with each user a list of other users the first trusts
  - Before executing any procedure, system checks that user executing procedure trusts procedure owner

Multilevel Policies

- Put programs at the lowest security level, all subjects at higher levels
  - By *-property, nothing can write to those programs
  - By ss-property, anything can read (and execute) those programs
- Example: Trusted Solaris system
  - All executables, trusted data stored below user region, so user applications cannot alter them

Proof-Carrying Code

- Code consumer (user) specifies safety requirement
- Code producer (author) generates proof code meets this requirement
  - Proof integrated with executable code
  - Changing the code invalidates proof
- Binary (code + proof) delivered to consumer
- Consumer validates proof
- Example statistics on Berkeley Packet Filter: proofs 300–900 bytes, validated in 0.3 –1.3 ms
  - Startup cost higher, runtime cost considerably shorter

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!
  - Or libraries …
- 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