Data Security and Privacy

Topic 7: Usable Integrity Protection

Readings

- Usable Mandatory Integrity Protection for Operating Systems
  - Ninghui Li, Ziqing Mao, and Hong Chen
    In *IEEE Symposium on Security and Privacy*, May 2007.
- Combining Discretionary Policy with Mandatory Information Flow in Operating Systems.
  - Ziqing Mao, Ninghui Li, Hong Chen, Xuxian Jiang:
  - ACM Trans. Inf. Syst. Secur. 14(3): 24:1-24:27(2011)

Motivation

- Host compromise by network-based attacks is the root cause of many serious security problems
  - Worm, Botnet, DDoS, Phishing, Spamming
- Why hosts can be easily compromised
  - Programs contain exploitable bugs
  - The discretionary access control mechanism in the operating systems was not designed to take buggy software in mind

Six design principles for usable access control systems <1>

- *Principle 1: Provide "good enough" security with a high level of usability; rather than "better" security with a low level of usability*
  - Need to trade off "theoretical security" for usability
- *Principle 2: Provide policy, not just mechanism*
  - Go against the UNIX "mechanism-but-not-policy" philosophy
- *Principle 3: Have a well-defined security objective*
  - Simplify policy specification while achieving the objective

Six design principles for usable access control systems <2>

- *Principle 4: Carefully design ways to support exceptions in the policy model*
  - Design exception mechanisms to the global MAC policy rules to minimize attack surface

- *Principle 5: Rather than trying to achieve "strict least privilege", aim for "good-enough least privilege"*
  - Aim also at minimizing policy specifications

- *Principle 6: Use familiar abstractions in policy specification interface*
  - Design for psychological acceptability

The UMIP Model: Security Objective

- Protect against network-based attacks
  - Network servers and client programs contain bugs
  - Users may make careless mistakes, e.g., downloading malicious software and running them
  - Attacker does not have physical access to the host
- The security property we want to achieve
  - The attacker cannot compromise the system integrity (except through limited channels)
    - E.g, install a RootKit, gain the root privileges
  - The attacker can get limited privileges
    - Run some code
  - After a reboot, the attacker does not present any more

The UMIP Model: Usability Objectives

- Easy policy configuration and deployment
- Understandable policy specification
- Nonintrusive: existing applications and common usage practices can still be used

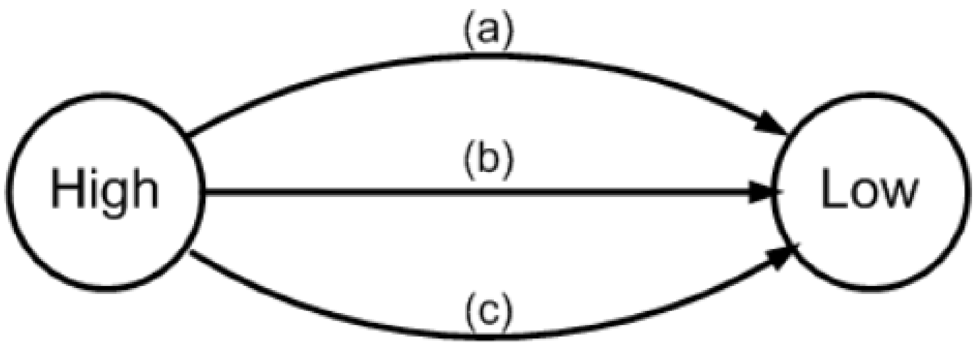
Basic UMIP Model

- Each process is associated with one bit to denote its integrity level, either high or low
  - A process having low integrity level might have been contaminated
- A low-integrity process by default cannot perform any sensitive operations that may compromise the system
- Three questions
  - How to do process integrity tracking?
  - What are sensitive operations?
  - What kinds of exceptions do we need?

Process Integrity Tracking

- Based on information flow

When a process is created, it inherits the parent's IL



The state-transition rules for processes:

(a): receive remote network traffic

(b): receive IPC traffic from a low-integrity process

(c): read a low-integrity file

File Integrity Tracking

- Non-directory files have integrity tracking
  - use the sticky bit to track whether a file has been contaminated by a low-integrity process
  - a file is low integrity if either it is not write-protected, or its sticky bit is set
  - the sticky bit can be reset by running a special utility program in high integrity
    - allow downloading and installing new programs

Sensitive Operations: Capabilities

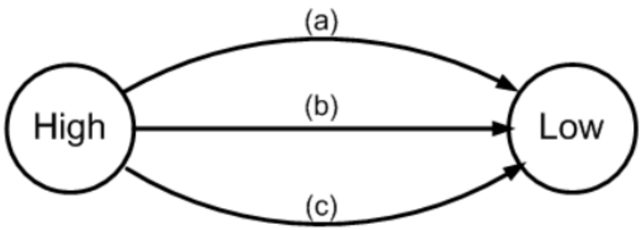
- Non-file sensitive operations
  - E.g., loading a kernel module, administration of IP firewall,…
- Using the Capability system
  - Break the root privileges down to smaller pieces
  - In Linux Kernel 2.6.11, 31 different capabilities
- Identify each capability as one kind of non-file sensitive operation

Sensitive Operations: File Access

- Asking users to label all files is a labor intensive and error-prone process
- Our Approach: Use DAC information to identify sensitive files
- Read-protected files
  - Owned by system accounts and not readable by world
  - E.g., /etc/shadow
- Write-protected files
  - Not writable by world
  - Including files owned by non-system accounts

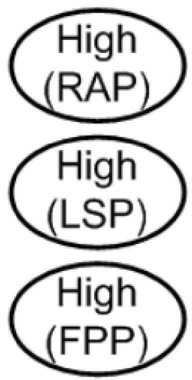
Exception Policies: Process Integrity Tracking

- Default policy for process integrity tracking



The state-transition rules for processes:
(a): receive remote network traffic
(b): receive IPC traffic from a low-integrity process
(c): read a low-integrity file

- Exceptions:



: maintain the integrity when (a) happens
: maintain the integrity when (b) happens
: maintain the integrity when (c) happens

- Examples
  - RAP programs: SSH Daemon
  - LSP programs: X server, desktop manager

Exception Policies: Low-integrity Processes Performing Sensitive Operations

- Some low-integrity processes need to perform sensitive operations normally
- Exception:

- Examples:
  - FTP Daemon Program: /usr/sbin/vsftpd
  - Use capabilities: CAP_NET_BIND_SERVICE, CAP_SYS_SETUID, CAP_SYS_SETGID, CAP_SYS_CHROOT
  - Read read-protected files: /etc/shadow
  - Write write-protected files: /etc/vsftpd, /var/log/xferlog

Implementation & Performance

- Implemented using Linux Security Module
  - no change to Linux file system
- Performance
  - Use the Lmbench 3 and the Unixbench 4.1 benchmarks
  - Overheads are less than 5% for most benchmark results

Part of the Sample Policy

Services and Path of the Binary	Type	File Exceptions	Capability Exceptions
SSH Daemon /usr/sbin/sshd	RAP		
Automated Update: /usr/bin/yum	RAP		
/usr/bin/vim	FPP		
/usr/bin/cat	FPP		
FTP Server /usr/sbin/vsftpd	NONE	(/var/log/xferlog, full) (/etc/vsftpd, full, R) (/etc/shadow, read)	CAP_SYS_CHROOT CAP_SYS_SETUID CAP_SYS_SETGID CAP_NET_BIND_SERVICE
Web Server /usr/sbin/httpd	NONE	(/var/log/httpd, full, R) (/etc/pki/tls, read, R) (/var/run/httpd.pid, full)	
Samba Server /usr/sbin/smbd	NONE	(/var/cache/samba, full, R) (/etc/samba, full, R) (/var/log/samba, full, R) (/var/run/smbd.pid, full)	CAP_SYS_RESOURCE CAP_SYS_SETUID CAP_SYS_SETGID CAP_NET_BIND_SERVICE CAP_DAC_OVERRIDE
NetBIOS name server /usr/sbin/nmbd	NONE	(/var/log/samba, full, R) (/var/cache/samba, full, R)	
Version control server /usr/bin/svnserve	NONE	(/usr/local/svn, full, R)	

Differences with Other Integrity Models

- Use multiple policies from the Biba model
  - subject low water for most subjects/processes
  - ring policy for some trusted subjects
    - e.g., ssh daemon, automatic update programs
  - object low water for some objects
- Each object has a separate protection level and integrity level
  - integrity level for quality information
  - protection level for important
    - read protection level inferred from DAC permissions on read
    - write protection level inferred from DAC permissions on write

Differences with Other Integrity Models

- Other exceptions to formal integrity rules
  - low integrity objects can be upgraded to high by a high integrity subject
  - low integrity subjects can access high protected objects via exceptions

Limitation of UMIP

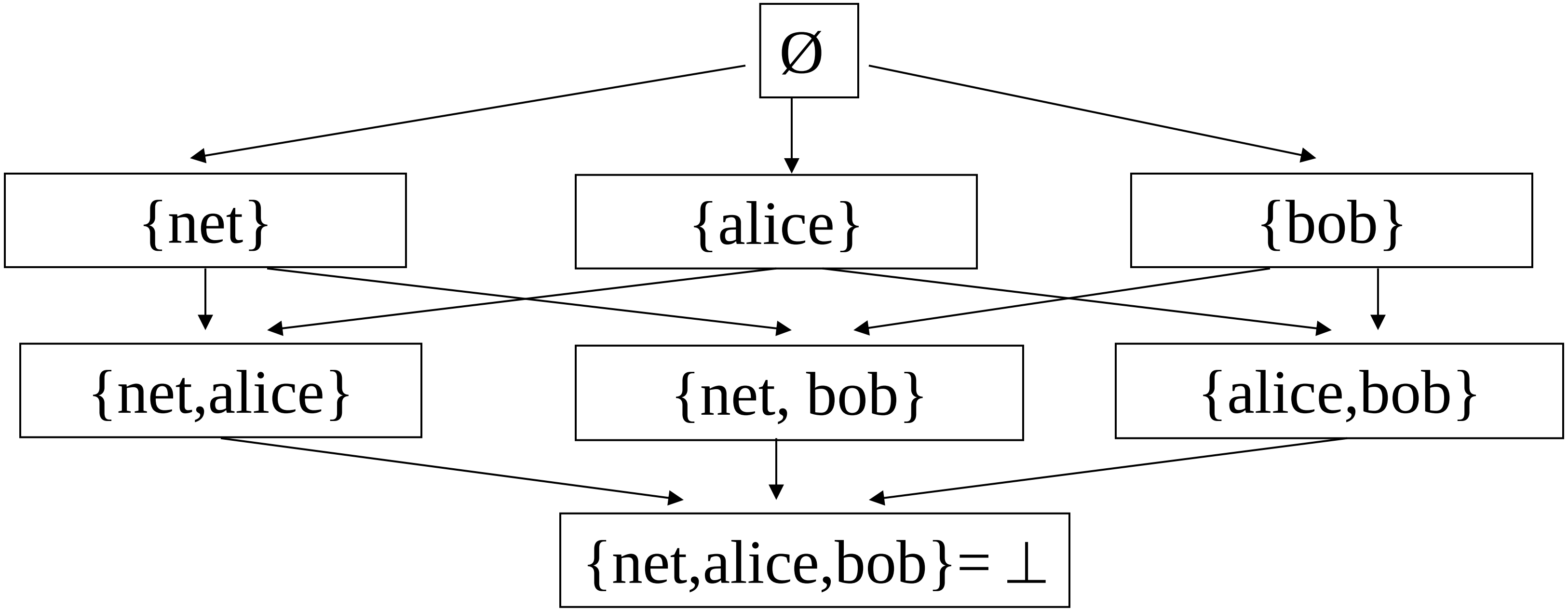
- Separates the system between network (low) and system critical (high)
- What to do with normal user files?
  - Treat them as low:
    - User files are not protected
  - Treat them at high
    - Malicious users (or users with weak passwords) lead to compromise of the protection
- Solution: Information Flow Enhanced Discretionary Access Control (IFEDAC)

Principals in IFEDAC

- An entity that may potentially compromise the system
- local users (DAC user accounts)
- Remote network traffic
  - denoted as net
  - represents the remote adversary

Integrity Levels in IFEDAC

- Maintain an integrity level for each process & file
  - A label is a a set of principals
  - E.g., {alice}, Ø, {bob, net}, {net}, …



Integrity Level

- For a process, the label contains principals
  - Who MAY have gained control over the process

- For a file, the label contains principals
  - who have changed the content stored in the file

Integrity Level Tracking

- Track integrity levels using information flow
  - p is newly created → assign p'parent.IL to p.IL
  - p receives network communication → add {net} to p.IL
  - p reads a file f → add f.IL to p.IL
  - p receives IPC data from p' → add p'.IL to p.IL
  - p creates a file f → assign p.IL to f.IL
  - p writes to a file f → add p.IL to f.IL
  - p logs in a user u → add {u} to p.IL
- Initial integrity level labeling
  - The first process init.IL = top (Ø)

Integrity Level Examples

- For example
  - Web server's IL = {net}
  - Alice's email client's IL = {net, Alice}
  - A file saved from Alice's email attachment has IL = {net, Alice}
  - pdf viewer's IL = {Alice}
  - pdf viewer's IL after opens an email attachment = {net, Alice}

File Protection Classes

- Each file has three protection classes
  - Read protection class (rpc): who can read it
  - Write protection class (wpc): who can write to it
  - Admin protection class (apc): who can change its rpc and wpc
  - Each value is a set of principals
- Infer file protection classes from DAC policy
  - f.rpc
    - If f is world-readable, f.rpc = $\perp$
    - Otherwise, f.rpc = the set of users allowed to read f
  - Same for wpc
  - f.apc = {owner}

IFEDAC Policy

- An access is allowed if all principals in the process’s IL are authorized
- A process p requests to access a file f
  - Allow reading, if p.IL $\subseteq$ f.rpc
  - Allow writing, if p.IL $\subseteq$ f.wpc
  - Allow changing f.rpc, f.wpc and f.apc, if p.IL $\subseteq$ f.apc
- File’s integrity level can be explicitly changed by user
  - Only the owner of the file can change a file’s integrity level, and only up to the int. level of the current process
    - I.e.,f.IL to IL’, if p.IL $\subseteq$ f.apc and p.IL $\subseteq$ IL’

Exceptions

- Default policy too strict for real-world systems and common practices
  - it doesn't assume any program to be correct
- In reality one has to trust the correctness of "some" program, needs exceptions to the default policy
- Exceptions are associated with program binaries
- Exceptions imply some form of trust for programs
  - The trusts are strictly limited and can be clearly specified

What Protection Does IFEDAC Offer?

- Achieve the protection objective of DAC, i.e., all allowed operations reflect the intention of authorized users, under the following assumptions
  - Initially, the inferred file integrity levels are correct
  - Initially, files are labeled with correct DAC policies
  - Hardware is not compromised
  - Kernel cannot be exploited in a critical way
  - When a legitimate user intends to upgrade a file's integrity level (or update a file's protection classes), the decision is correct
  - Exceptions are justified

Usage Case I: Email Client (cont')

- John saves an email attachment B to /home/john/download
  - B.IL = {john, net}
- John wants to install B to the system, so executes B as BP
  - BP.IL = {john, net}
  - BP cannot touch the system files, installation failed if needs such access
  - BP cannot access files that are not world accessible (can change contents of B's Internet directory)
- John really trusts B and wants to install it
  - John login as an administrator (see below)
  - John explicitly upgrades B.IL to top
- John executes B as BP'
  - BP'.IL = top, installation succeed

Usage Case II: Administrator Login

- Linux allows normal users to perform system administration through the sudo tool (sudoer)
- IFEDAC allows specifying privileged users, called sudoers
  - Process's IL maintains when a sudoer logins
- Sudoers' files have wpc at {u} or lower
  - Except the shell startup scripts with wpc at top
    - .bash_rc, .bash_profile, .bash_history
- When a sudoer John logins
  - John gets a shell with IL at top
  - John can perform system administration in the shell
  - Any descendant that reads john's normal files will drop to IL {john}
  - A utility program is provided to explicitly downgrade shell's IL to {john}

Comparing IFEDAC with Biba (1)

- In Biba, an object has one integrity level
  - Determines who can write to it, and how will it contaminates a subject who reads
- In IFEDAC, an object has
  - An integrity level, records quality of info in the object, and ensures correct contamination tracking
  - A write protection class, determines who can write it and protects integrity of the object
  - A read protection class, determines who can read it and protects confidentiality of the object
- IFEDAC infers protection classes from DAC permissions

Comparing IFEDAC with Biba

- IFEDAC uses aspects of all five Biba policies
  - Subject low water policy for majority of subjects
  - Ring policy for selected subjects (i.e., RAP & LSP, which are explicitly identifying trusted programs)
  - Object low water policy when objects has low write protection class (e.g., temporary files)
  - Strict integrity for objects that have high write protection class (e.g., critical binaries and configuration files)
  - Strict integrity protection for subject-subject interaction

Summary of IFEDAC

- DAC's weakness lies in the enforcement
  - The origin includes a single principal
  - Failed to identify the true origins of a request
  - Vulnerable to Trojan horse and buggy software
- But DAC's policy is good
  - Easy and intuitive to specify
  - Sufficient to preserve the system integrity
- The approach
  - Keep the DAC's policy
  - Fix the enforcement: identify the true origins of a request

Coming Attractions …

- Role Based Access Control