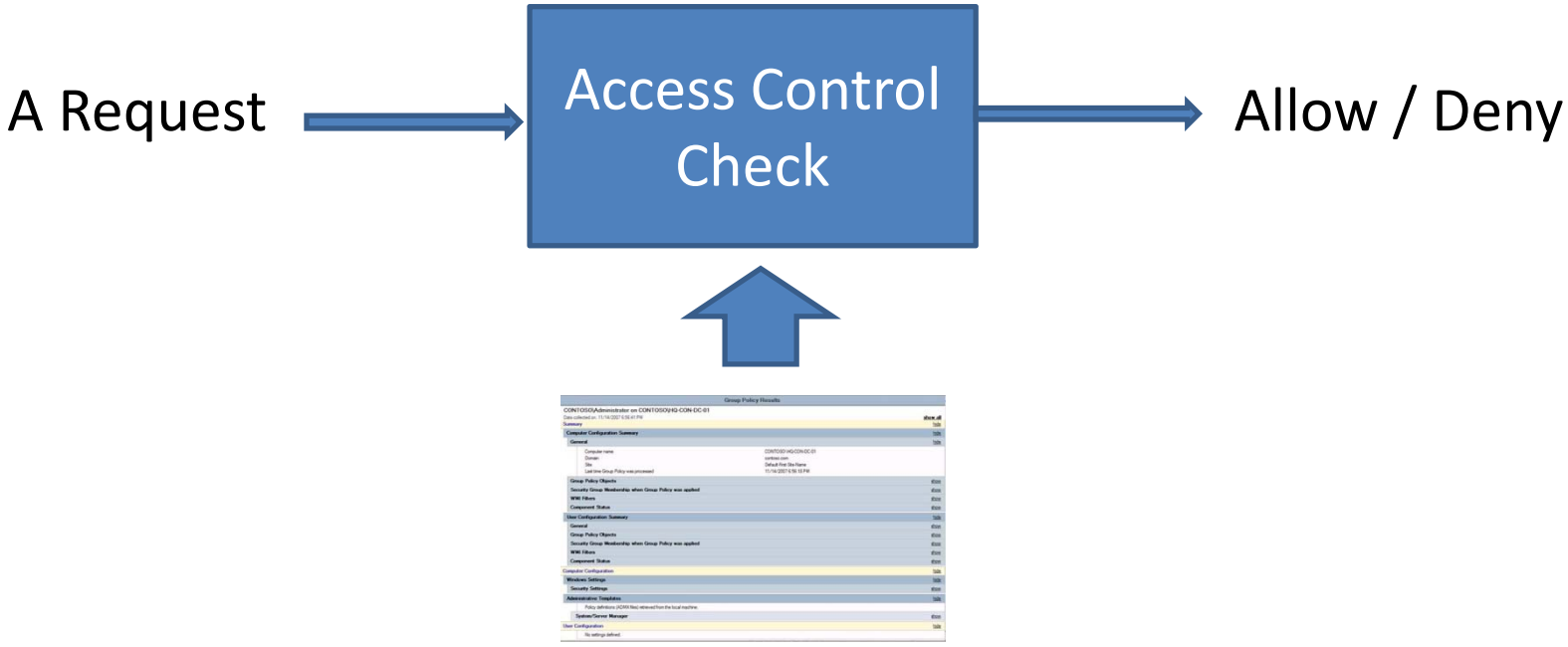
# Information Security
# CS 526
## Lecture 20

## UMIP & IFEDAC

# Access Control Check

- Given an access request, return an access control decision based on the policy
  - allow / deny



# Access Request

- Examples
  - Operating system: process A wants to read file B
  - Browser: the script in webpage X wants to access webpage Y
- A request
  - Subject: the entity who issues the request
    - A piece of code
    - E.g., a process in operating systems, a webpage in browsers
  - Action: object + access mode
- Check whether the subject has the privilege to perform the action, based on the policy

# Policy

- Grant privileges to principals
- Examples of principals
  - Operating system: user accounts

# The Gap

- A request: a subject wants to perform an action
- The policy: each principal has a set of privileges
- To fill the gap between the subjects and the principals
  - relate the subject to the principals

# Origin

- The origins of a request
  - The set of principals on whose behalf the subject is executing at the time when the request is issued
- Origin links the subjects with the principals
  - The origins cause the subject to issue the request
  - The origins should be responsible for the request
- Check whether the origins have the privilege to perform the action based on the policy

# Identifying the Origins

- Properly identifying the origins is critical to making correct access control decisions
- Some real-world access control systems are vulnerable because of their limitations in identifying the origins
  - Incompleteness
- Examine two real-world access control systems
  - DAC in the operating system
  - SoP in the browser
- Propose enhancements

# Discretionary Access Control (DAC)

- Implemented in almost all modern operating systems

- No precise definition
- Features
  - Restrict access based on the identity of subjects
  - Each object has a unique owner
  - Owner decides who can access

# Origins in DAC

- Principals
  - User accounts
- The origin of a request
  - The user account who starts the subject process
  - E.g., euid in UNIX
  - invoker

# DAC is NOT enough

- Vulnerable to Trojan Horses
  - Malicious software
- When a user executes a Trojan program
  - The origin in DAC: the victim user
  - The process (adversary) gains the victim user's privileges

# DAC is NOT enough (cont')

- Vulnerable to buggy software
  - The adversary can exploit the vulnerability and inject malicious code
- When a program is exploited
  - The origin is the invoker
  - The injected code gains the invoker's privileges
- Server daemons are attractive targets
  - commonly vulnerable
  - Started by administrator

# Why DAC is vulnerable?

- Implicit assumptions
  - Software are benign, i.e., behave as intended
  - Software are correct, i.e., bug-free
- The reality
  - Malware are popular
  - Software are vulnerable

# Why DAC is Vulnerable? (cont')

- Fundamental reason
  - A single invoker is not enough to capture the origins of a process
- When the program is a Trojan
  - The program-provider should be responsible for the requests
- When the program is vulnerable
  - It may be exploited by input-providers
  - The requests may be issued by injected code from input-providers

# Usable Mandatory Integrity Protection (UMIP)

- Preserve host integrity against the remote adversary
  - Remote exploitation
  - Downloaded malware
- Idea:  add an integrity level to the origin of a process
  - Either high or low
- Low integrity means the process may have been contaminated
  - be exploited by remote attackers
  - executing a downloaded malware

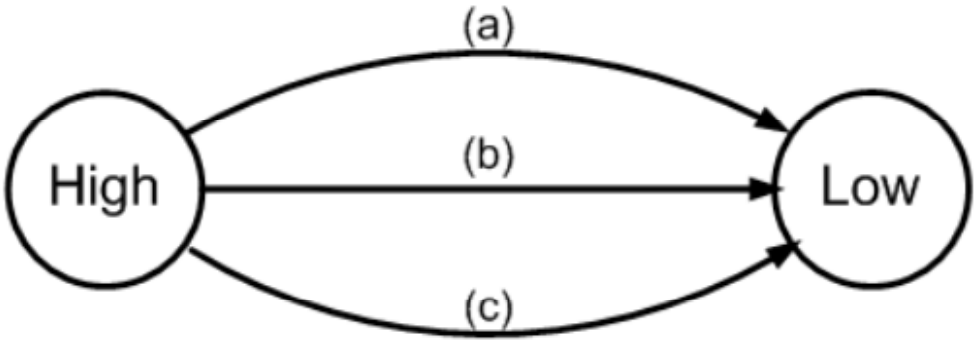
# Basic UMIP Model

- 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

# Sensitive Operations: File Access

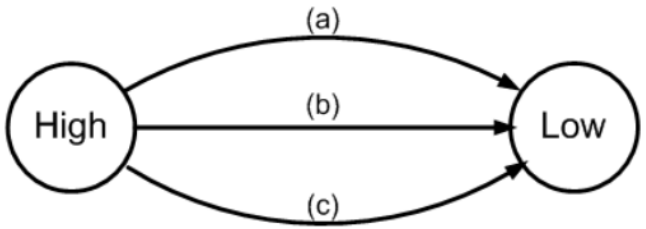
- Asking users to label all files is a labor intensive and error-prone process
- Our Approach: Use DAC information to identify sensitive files
- Read-protected files
  - Owned by system accounts and not readable by world
  - E.g., /etc/shadow
- Write-protected files
  - Not writable by world
  - Including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

# Sensitive Operations: Capabilities

- Sensitive non-file 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 2.6.20, 31 different capabilities
- Identify each capability as one kind of sensitive non-file operation

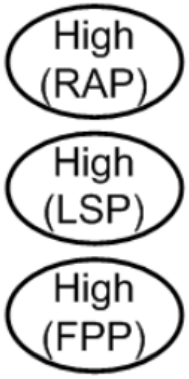
# Exception Policies: Process Integrity Tracking

- 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
  - FPP programs: vim, cat

# Exception Policies: Low-integrity Processes Performing Sensitive Operations

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


: can do operations allowed by special privileges

- 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

# Usability Evaluation

- Experimental Settings
  - Establish a server with Fedora Core 5
  - Enable UMIP implementation during system boot
  - Install several commonly used services
    - E.g., http, ftp, samba, svn, smtp, ntp, …
- Results
  - The system works with a small and easily-understood policy
  - The system has been used by our group for about one year
  - With the policy, remote administration through SSH and X administration are enabled

# Evaluation: 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)<br>(/etc/vsftpd, full, R)<br>(/etc/shadow, read) | CAP_SYS_CHROOT<br>CAP_SYS_SETUID<br>CAP_SYS_SETGID<br>CAP_NET_BIND_SERVICE |
| Web Server /usr/sbin/httpd | NONE | (/var/log/httpd, full, R)<br>(/etc/pki/tls, read, R)<br>(/var/run/httpd.pid, full) | |
| Samba Server /usr/sbin/smbd | NONE | (/var/cache/samba, full, R)<br>(/etc/samba, full, R)<br>(/var/log/samba, full, R)<br>(/var/run/smbd.pid, full) | CAP_SYS_RESOURCE<br>CAP_SYS_SETUID<br>CAP_SYS_SETGID<br>CAP_NET_BIND_SERVICE<br>CAP_DAC_OVERRIDE |
| NetBIOS name server /usr/sbin/nmbd | NONE | (/var/log/samba, full, R)<br>(/var/cache/samba, full, R) | |
| Version control server /usr/bin/svnserve | NONE | (/usr/local/svn, full, R) | |

# Limitations of UMIP

- Policy Model in UMIP
  - Trust the local users
  - Defeat remote attackers
- Local users are not trustworthy
  - Multi-user systems
  - User may exploit vulnerabilities

# Revisit: The Origins of a Process

- DAC
  - Origin: the invoker
- Who may control a process?
  - Invoker
  - Program provider
  - Input provider
- UMIP
  - Add the program-provider and input-providers to the origins
  - High / Low: whether it comes from network or has received network input
- Information Flow Enhanced DAC

# Principal

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

# Integrity Level

- Maintain an integrity level for each process and file
  - a set of principals
  - E.g., {alice}, Ø, {bob, net}, {net}, …
- For process
  - Who MAY have gained control over the process
- For file
  - Who have changed the content stored in the file
- The principals in the integrity level are the origins of a process

# Integrity Level Tracking

- Track integrity levels using information flow
- Rules
  - 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 = bot
    - 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
    - up to the integrity level of the current process
  - Allow changing f.IL to IL', if p.IL $\subseteq$ f.apc and p.IL $\subseteq$ IL'

# 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

# IFEDAC Policy Enhances DAC Policy

- Generally, IFEDAC enforces DAC policy
  - A protection class contains the authorized entities in a DAC policy
- The protection classes can be different from that inferred from DAC
  - Stored using extended attributes
  - Can be explicitly set by users
- Two example cases
  - Internet directory: DAC (only owner can write), wpc = {u, net}
  - The shell startup scripts of privileged normal users:
    DAC (writable by owner), wpc= top

# 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

# Usage Case I: Email Client (cont')

- John saves another 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 UMIP & IFEDAC with Biba (2)

- In Biba, an object has one integrity level
  - Determines who can write to it, and how will it contaminates a subject who reads
- In UMIP & 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 UMIP & IFEDAC with Biba (3)

- UMIP and 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

- Trusted Operating Systems and Assurance