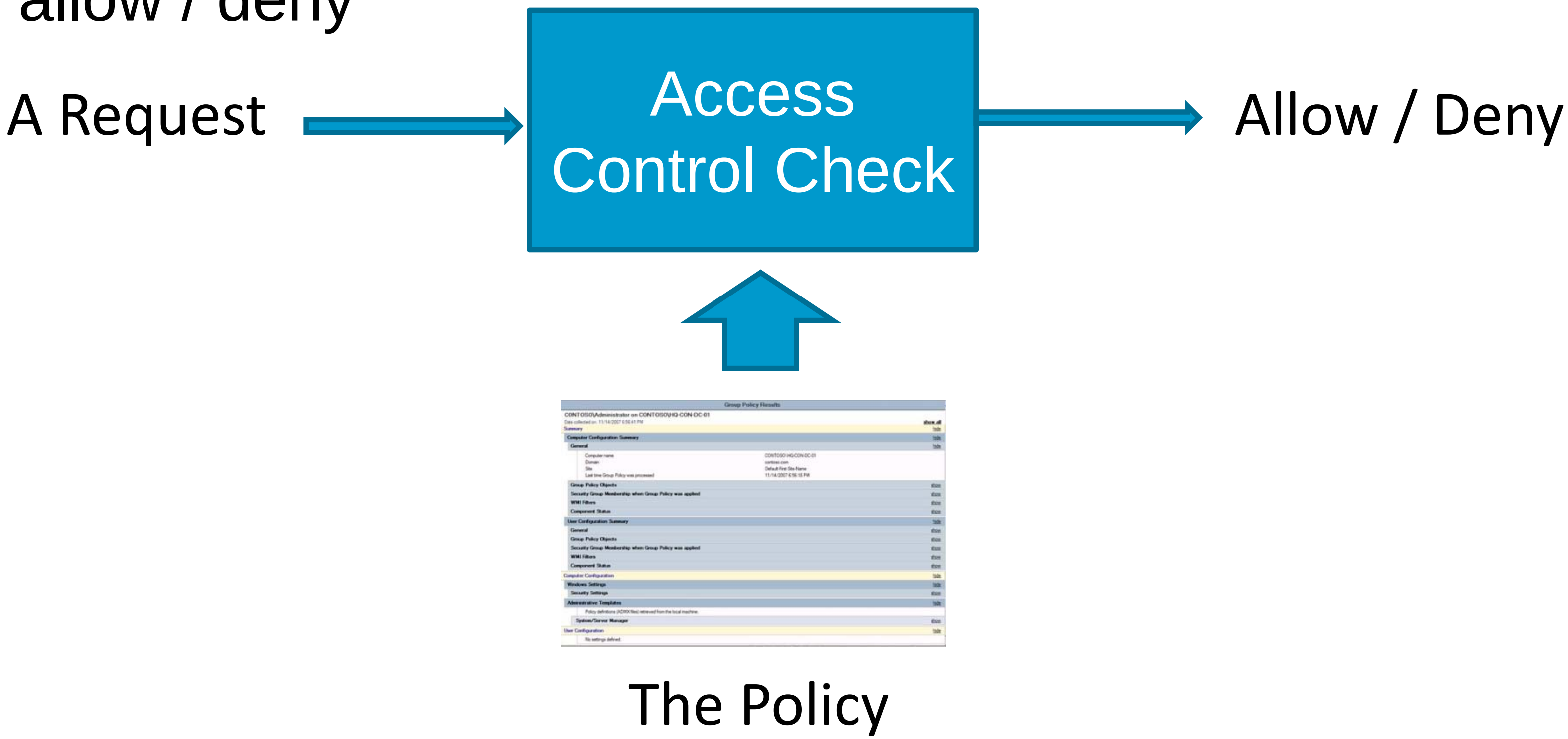
Computer Security
CS 426
Lecture 29

IFEDAC & Windows MIC

Access Control Check

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



The Gap Between Request & Policy

- A request: a subject wants to perform an action
  - E.g., processes in OS
- The policy: each principal has a set of privileges
  - E.g., user accounts in OS

- Challenging to fill the gap between the subjects and the principals
  - relate the subject to the principals

Unix DAC Revisited (1)

Action	Process	Effective UID	Real Principals
User A Logs In	shell	User A	User A
Load Binary "Goodie" Controlled by user B	Goodie	User A	? ?

- When the Goodie process issues a request, what principal(s) is/are responsible for the request?
- Under what assumption, it is correct to say that User A is responsible for the request?

Assumption: Programs are benign, i.e., they only do what they are told to do.

UNIX DAC Revisited (2)

Action	Process	Effective UID	Real Principals
	shell	User A	User A
Load AcroBat Reader Binary	AcroBat	User A	User A
Read File Downloaded from Network	AcroBat	User A	? ?

- When the AcroBat process (after reading the file) issues a request, which principal(s) is/are responsible for the request?
- Under what assumption, it is correct to say that User A is responsible for the request?

  Assumption: Programs are correct, i.e., they handle inputs correctly.

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
- The problem is not caused by the discretionary nature of policy specification!
  – i.e., owners can set policies for files

Why DAC is Vulnerable? (cont')

- A deeper reason in the enforcement mechanism
  - 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

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

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)

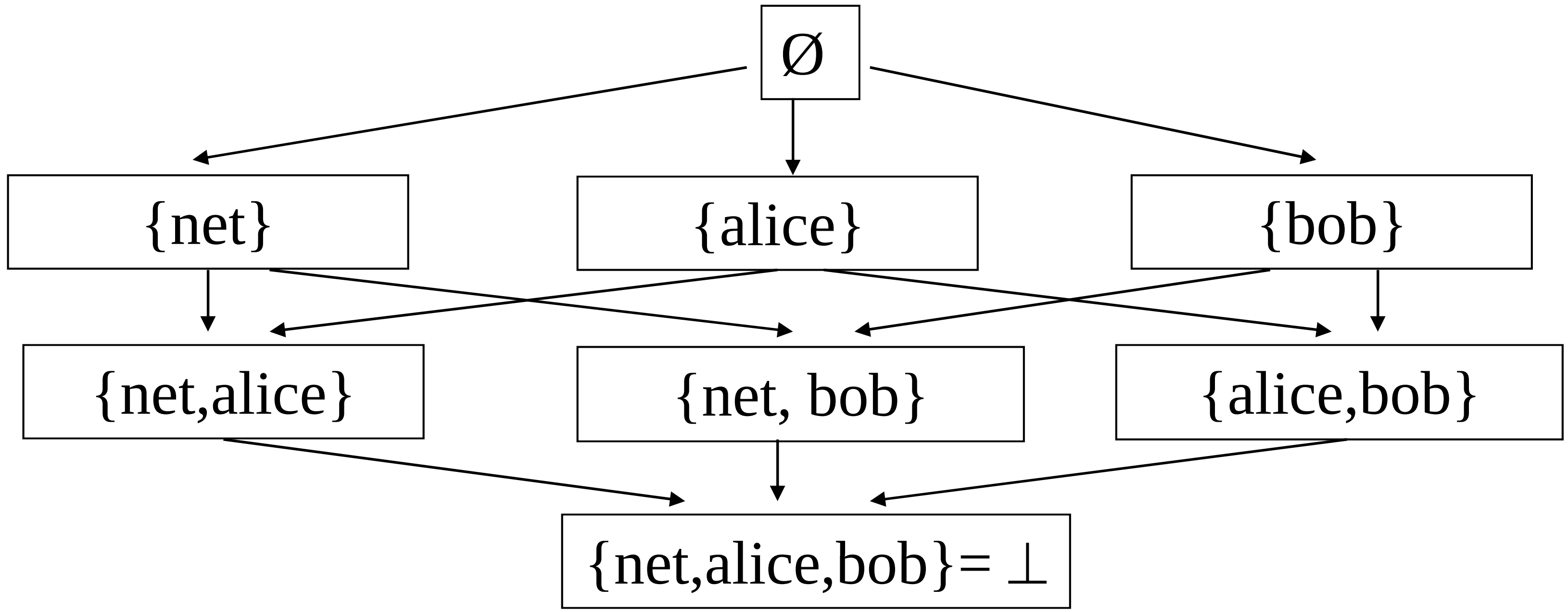
IFEDAC Overview

- Key Idea of IFEDAC:
  - Maintains a set of principals that could be responsible for any request
  - A request is authorized if all principals in the responsible set are authorized

- Principals in IFEDAC: Entities 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

Windows Mandatory Integrity Control

- Security feature since Vista
- Motivated by Biba
- Four integrity levels are used:
  - Low, medium, high, system
- Each process has an integrity level
  - Process starts with medium by default
  - Can get high with User Account Control
  - Process can be configured to start as low (such as browsers in protected mode)
    - What they can do are greatly limited

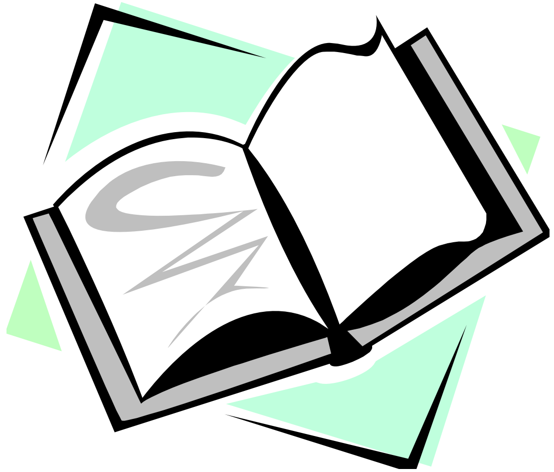
Windows Mandatory Integrity Control

- Each protected objects (files, registry keys) can specify the minimal integrity level for updating

- No dynamic information flow tracking
  - Even low-integrity can save files to exploit

Readings for This Lecture

- Optional:
  - Mao et al.: “Trojan Horse Resistant Discretionary Access Control” in SACMAT 2009.



Coming Attractions …

- Role Based Access Control