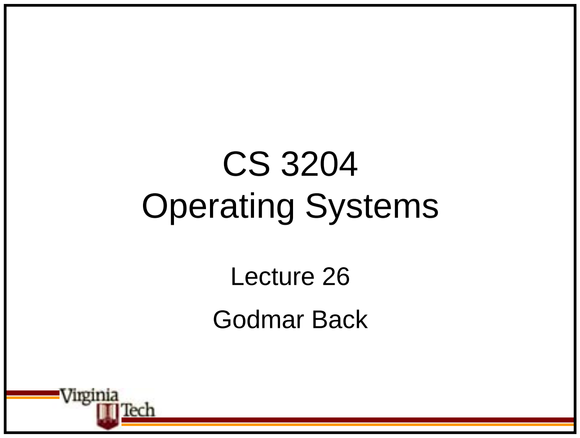
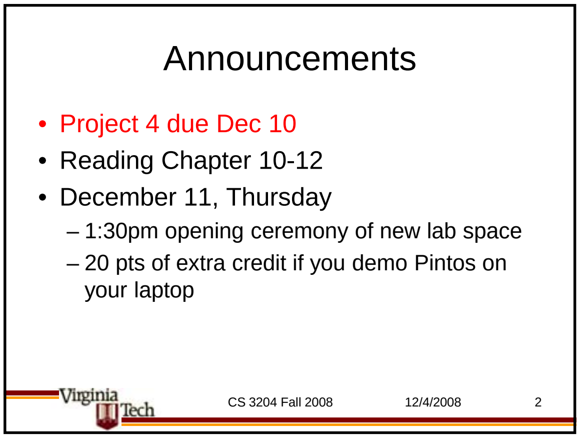


CS 3204 Operating Systems

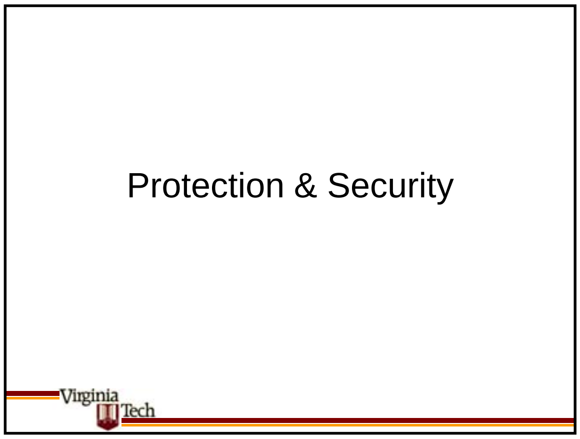
Lecture 26

Godmar Back

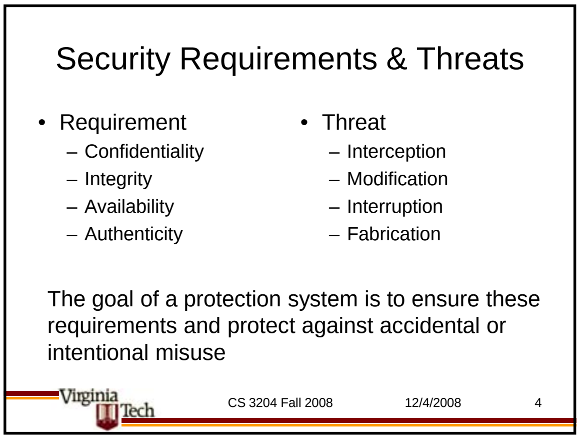


Announcements

- Project 4 due Dec 10
- Reading Chapter 10-12
- December 11, Thursday
  - 1:30pm opening ceremony of new lab space
  - 20 pts of extra credit if you demo Pintos on your laptop



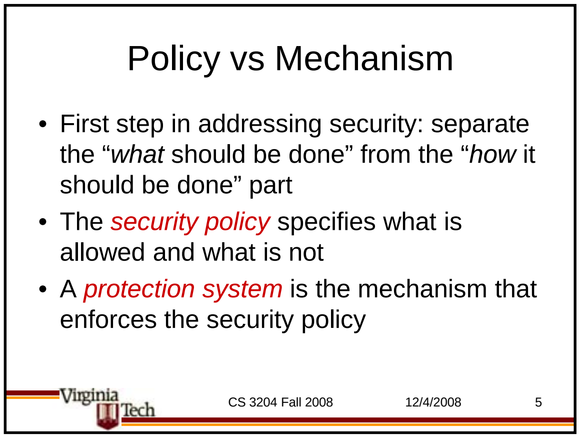
Protection & Security



Security Requirements & Threats

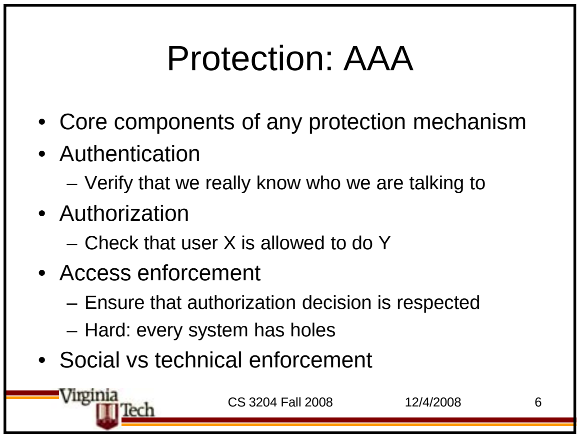
- Requirement
  - Confidentiality
  - Integrity
  - Availability
  - Authenticity
- Threat
  - Interception
  - Modification
  - Interruption
  - Fabrication

The goal of a protection system is to ensure these requirements and protect against accidental or intentional misuse



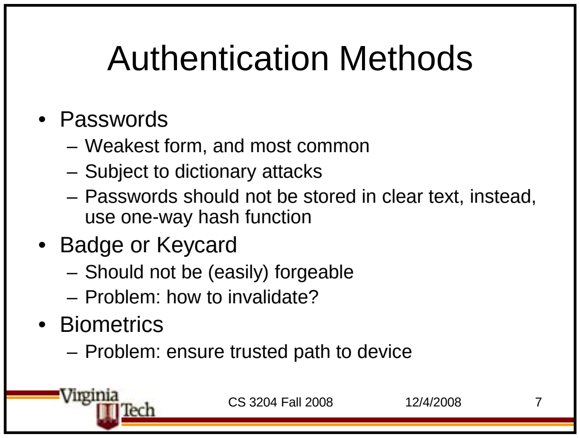
Policy vs Mechanism

- First step in addressing security: separate the “*what* should be done” from the “*how* it should be done” part
- The *security policy* specifies what is allowed and what is not
- A *protection system* is the mechanism that enforces the security policy



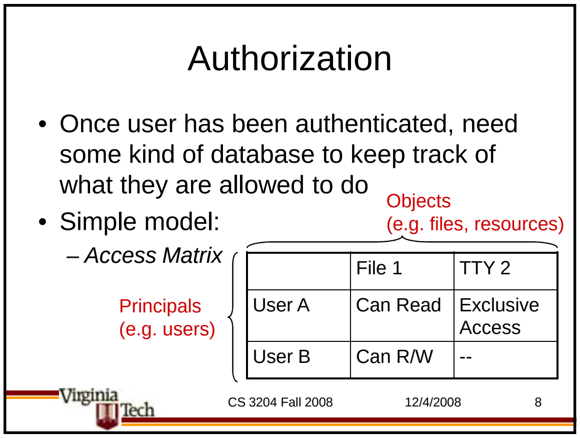
Protection: AAA

- Core components of any protection mechanism
- Authentication
  - Verify that we really know who we are talking to
- Authorization
  - Check that user X is allowed to do Y
- Access enforcement
  - Ensure that authorization decision is respected
  - Hard: every system has holes
- Social vs technical enforcement



Authentication Methods

- Passwords
  - Weakest form, and most common
  - Subject to dictionary attacks
  - Passwords should not be stored in clear text, instead, use one-way hash function
- Badge or Keycard
  - Should not be (easily) forgeable
  - Problem: how to invalidate?
- Biometrics
  - Problem: ensure trusted path to device



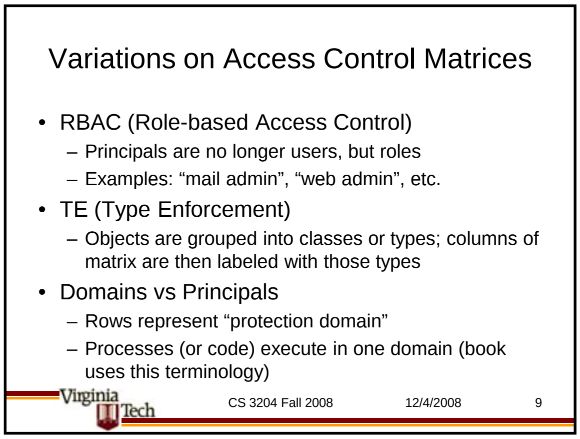
Authorization

- Once user has been authenticated, need some kind of database to keep track of what they are allowed to do
- Simple model:
  - *Access Matrix*

Objects (e.g. files, resources)

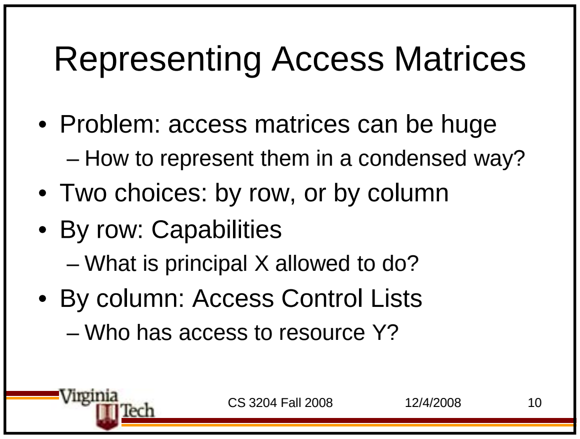
Principals (e.g. users)

	File 1	TTY 2
User A	Can Read	Exclusive Access
User B	Can R/W	--



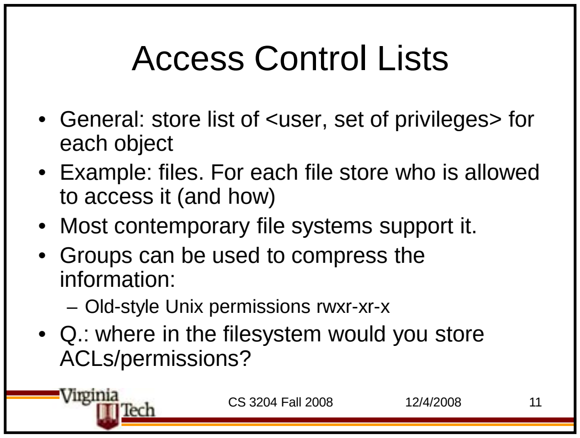
Variations on Access Control Matrices

- RBAC (Role-based Access Control)
  - Principals are no longer users, but roles
  - Examples: “mail admin”, “web admin”, etc.
- TE (Type Enforcement)
  - Objects are grouped into classes or types; columns of matrix are then labeled with those types
- Domains vs Principals
  - Rows represent “protection domain”
  - Processes (or code) execute in one domain (book uses this terminology)



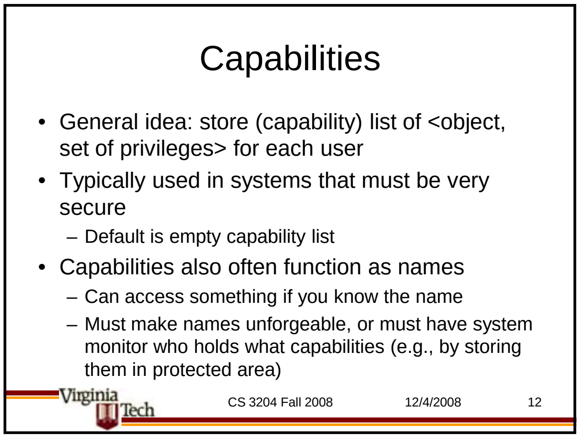
Representing Access Matrices

- Problem: access matrices can be huge
  - How to represent them in a condensed way?
- Two choices: by row, or by column
- By row: Capabilities
  - What is principal X allowed to do?
- By column: Access Control Lists
  - Who has access to resource Y?



Access Control Lists

- General: store list of <user, set of privileges> for each object
- Example: files. For each file store who is allowed to access it (and how)
- Most contemporary file systems support it.
- Groups can be used to compress the information:
  - Old-style Unix permissions rwxr-xr-x
- Q.: where in the filesystem would you store ACLs/permissions?



Capabilities

- General idea: store (capability) list of <object, set of privileges> for each user
- Typically used in systems that must be very secure
  - Default is empty capability list
- Capabilities also often function as names
  - Can access something if you know the name
  - Must make names unforgeable, or must have system monitor who holds what capabilities (e.g., by storing them in protected area)

Examples of Attacks

- Abuse of valid privilege
  - Admin decides to delete your mp3s
- Denial of service attack
  - Run this loop on your P4:
    - while (1) { mkdir(“x”); chdir(“x”); }
- Sniffing/Listening attack
- Trojan Horse
- Worm or virus

Simple Stack Overflow Example

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int
unsafe_function()
{
   char buf[8];
   printf("Enter your name: ");
   gets(buf);
   printf("Your name is: %s\n", buf);
}

void
do_something_bad()
{
   printf("do_something_bad called!\n");
}

int
main()
{
   unsafe_function();
}
```

```
> > nm stackattack | grep do_something_bad
08048456 T do_something_bad
>  od -h badinput
0000000 8456 0804 8456 0804 8456 0804 8456 0804
*
0000120
./stackattack < badinput
Enter your name: Your name is: V
V
do_something_bad called!
do_something_bad called!
```

- Simplified variant of “return-to-libc” attack in which attacker redirects control to function embedded in program (which then compromises security)
- In practice, attacker usually sends position-independent code along that exec()’s a shell

Defense against Stack Overflow

- Static Defenses:
  - Use of type-safe languages
  - Code analysis
- Dynamic Defenses:
  - Detect stack corruption before abnormal control transfer occurs
    - Canaries, stack frame reallocation
    - -fstack-protector in gcc (ProPolice)
  - Prevent execution of any code on stack
    - Remove execute privilege from stack
    - Generalization: W⊗X
  - Address Space Randomization
    - Helps against attacks that require that attacker knows address in program (less effective on 32-bit systems)

Principle of Least Privilege

- Containment
- “need-to-know” basis: every process should have only access right for the operations it needs to do its work
  - Hard to implement:
    - how can you be sure the program will still work? How can you be sure you’ve given just enough privileges and no more?
  - Example: Linux SE

Other Countermeasures

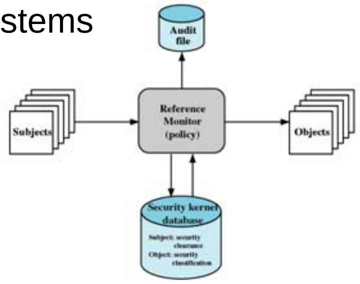
- Logging:
  - Keep an audit log of all actions performed
  - Must protect log (from theft & forgery)
- Verification & Proofs
  - Problem of verifying the specification vs. implementation

Trusted Computing Base

- The part of the system that enforces access control decisions
  - Also protects authentication information
- Issues:
  - Single Bug in TCB may compromise entire security policy
  - Need to keep it small and manageable
  - Usually: entire kernel is part of TCB (huge!)
    - Weakest link phenomenon

Trusted Systems

- MLS-Multilevel Security
  - Unclassified
  - Confidential
  - Secret
  - Top Secret
- No read up
- No write down
  - *-property


Audit file
Subjects
Reference Monitor (policy)
Objects
Security kernel database
Subject: security clearance
Object: security classification

Properties:

- Complete mediation (mandatory access control on every access)
- Isolated/tamper-proof reference monitor
- Verification (the hardest)

Security & System Structure

- Q.: Does system structure matter when building secure systems?
- Monolithic kernels: processes call into kernel to obtain services (Pintos, Linux, Windows)
- Microkernels: processes call only into kernel to send/receive messages, they communicate with other processes to obtain services
  - Asbestos [SOSP'05] exploits this to track information flow across processes
  - HiStar [OSDI'06] optimizes this further by avoiding explicit message passing; using “call gates” instead

Language-Based Protection

- Based on type-safe languages (Java, C#, etc.)
  - Do not allow direct memory access
  - Include access modifiers (private/public, etc.)
  - Verify code before they execute it with respect to these safety property
- Build security systems on top of type-safe language runtimes which associate code with sets of privileges