# Lecture 3
# September 29, 2025

ECS 235A, Computer and Information Security

# Administrative Stuff

- Slides for first 2 lectures now posted on Canvas
  - Go to the "Lecture Outlines" page for the links
  - They are in PDF
- Slides for this lecture are posted too
  - But I will update them after class

# Question

- A student's programming assignment was missing a file
- The TA asked the student to submit the missing file
- The student told the TA to get it from a given directory
- The TA found two files, identical in content except for the name of the programmer (which was in the file)
- The TA reported the student for cheating
  - Copying homework and submitting it as your ownb is, of course, cheating
- The student reported the TA for snooping
  - Snooping violated the University policy on privacy

# What Happened?

- To the student:
  - Committee ruled student cheated
  - First offense: suspended for the term
- To the TA:
  - Committee ruled no violation of policy as student told TA to find the file in the directory, so the TA *had* to look at contents of files to find the right one
- To the rules:
  - University policy on privacy revised to make clear that when an instructor (including TAs, graders, and others) is asked to look for a homework file, they are authorized to look at contents of files to find the right file

# Security Policy

- Policy partitions system states into:
  - Authorized (secure)
    - These are states the system can enter
  - Unauthorized (nonsecure)
    - If the system enters any of these states, it's a security violation
- Secure system
  - Starts in authorized state
  - Never enters unauthorized state

# Confidentiality

- *X* set of entities, *I* information
- *I* has the *confidentiality* property with respect to *X* if no $x \in X$ can obtain information from *I*
- *I* can be disclosed to others
- Example:
  - *X* set of students
  - *I* final exam answer key
  - *I* is confidential with respect to *X* if students cannot obtain final exam answer key

# Integrity

- *X* set of entities, *I* information
- *I* has the *integrity* property with respect to *X* if all $x \in X$ trust information in *I*
- Types of integrity:
  - Trust *I*, its conveyance and protection (data integrity)
  - *I* information about origin of something or an identity (origin integrity, authentication)
  - *I* resource: means resource functions as it should (assurance)

# Availability

- *X* set of entities, *I* resource
- *I* has the *availability* property with respect to *X* if all $x \in X$ can access *I*
- Types of availability:
  - Traditional: *x* gets access or not
  - Quality of service: promised a level of access (for example, a specific level of bandwidth); *x* meets it or not, even though some access is achieved

# Question

- University policy disallows cheating
  This includes copying homework, with or without permission
- CS class has students do homework on computer
- Anne forgets to read-protect her homework file
- Bill copies it
- Who breached security?
  - Anne, Bill, or both?

# Answer Part 1

- Bill clearly breached security
  - Policy forbids copying homework assignment
  - Bill did it
  - System entered unauthorized state (Bill having a copy of Anne's assignment)

- If not explicit in computer security policy, certainly implicit
  - Not credible that a unit of the university allows something that the university as a whole forbids, unless the unit explicitly says so

# Answer Part 2

- Security policy does not require users to protect their files
- Anne didn't protect her homework
  - Not required by the security policy
- She didn't breach security

- Security policy requires users to protect their files
- Anne didn't protect her homework
  - But this ***is*** required by the security policy (and University policy and rules)
- She breached security

# Policy Models

- Abstract description of a policy or class of policies
- Focus on points of interest in policies
  - Security levels in multilevel security models
  - Separation of duty in Clark-Wilson model
  - Conflict of interest in Chinese Wall model

# Mechanisms

- Entity or procedure that enforces some part of the security policy
  - Access controls (like bits to prevent someone from reading a homework file)
  - Disallowing people from bringing CDs and floppy disks into a computer facility to control what is placed on systems

# Entities

- Subject: active entity
  - Causes information to flow or system state to change
  - Examples: processes, some devices
  - At a higher layer of abstraction: users, other computers
- Object: passive entity
  - Contains or receives information
  - Examples: files, some devices
  - At a higher layer of abstraction: file server, network

# Types of Security Policies

- Military (governmental) security policy
  - Policy primarily protecting confidentiality
- Commercial security policy
  - Policy primarily protecting integrity
- Confidentiality policy
  - Policy protecting only confidentiality
- Integrity policy
  - Policy protecting only integrity

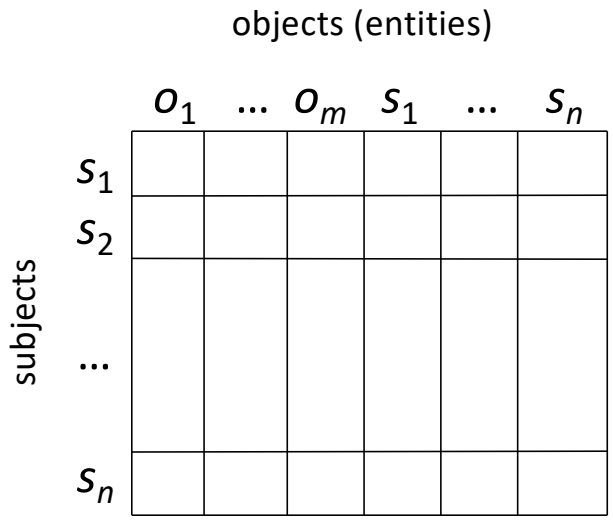
# Types of Access Control

- Discretionary Access Control (DAC, IBAC)
  - individual user sets access control mechanism to allow or deny access to an object
- Mandatory Access Control (MAC)
  - system mechanism controls access to object, and individual cannot alter that access
- Originator Controlled Access Control (ORCON)
  - originator (creator) of information controls who can access information

# Access Control Matrix

- Access Control Matrix Model
- Protection State Transitions
  - Commands
  - Conditional Commands
- Special Rights
- Principle of Attenuation of Privilege

# Description




- Subjects $S = \{ s_1, \ldots, s_n \}$
- Objects $O = \{ o_1, \ldots, o_m \}$
- Rights $R = \{ r_1, \ldots, r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject $s_i$ has rights $r_x, \ldots, r_y$ over object $o_j$

# Example 1

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

| | *f* | *g* | *p* | *q* |
|---|---|---|---|---|
| *p* | *rwo* | *r* | *rwxo* | *w* |
| *q* | *a* | *ro* | *r* | *rwxo* |

# Example 2

- Host names *telegraph*, *nob*, *toadflax*
- Rights *own*, *ftp*, *nfs*, *mail*

| | *telegraph* | *nob* | *toadflax* |
|---|---|---|---|
| *telegraph* | *own* | *ftp* | *ftp* |
| *nob* | | *ftp, mail, nfs, own* | *ftp, nfs, mail* |
| *toadflax* | | *ftp, mail* | *ftp, mail, nfs, own* |

# Example 3

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights *+*, *–*, *call*

| | *counter* | *inc_ctr* | *dec_ctr* | *manage* |
|---|---|---|---|---|
| *inc_ctr* | *+* | | | |
| *dec_ctr* | *–* | | | |
| *manager* | | *call* | *call* | *call* |

# State Transitions

- Change the protection state of system
  - Protection state is the triple ($S$, $O$, $A$), where S is the set of subjects, $O$ is the set of entities (not the set of passive entities, so S $\subseteq$ O) and $A$ is the access control matrix
- |– represents transition
  - $X_i$ |–$_\tau$ $X_{i+1}$: command $\tau$ moves system from state $X_i$ to $X_{i+1}$
  - $X_i$ |–$^*$ $Y$: a sequence of commands moves system from state $X_i$ to $Y$
- Commands often called *transformation procedures*

# Primitive Operations

- **create subject** *s*; **create object** *o*
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject** *s*; **destroy object** *o*
  - Deletes row, column from ACM; deletes column from ACM
- **enter** *r* **into** *A*[*s*, *o*]
  - Adds *r* rights for subject *s* over object *o*
- **delete** *r* **from** *A*[*s*, *o*]
  - Removes *r* rights from subject *s* over object *o*

# Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create•file(p, f)
    create object f;
    enter own into A[p, f];
    enter r into A[p, f];
    enter w into A[p, f];
end
```

# Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
    enter own into A[p, g];
end
```

- Mono-operational command
  - Single primitive operation in this command

# Conditional Commands

- Let *p* give *q* *r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
    if own in A[p, f]
    then
        enter r into A[q, f];
end
```

- Mono-conditional command
  - Single condition in this command

# Multiple Conditions

- Let *p* give *q* *r* and *w* rights over *f*, if *p* owns *f* and *p* has *c* rights over *q*

```
command grant•read•file•2(p, f, q)
    if own in A[p, f] and c in A[p, q]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
```