L2: Access Control Matrix

Hui Chen, Ph.D.
Dept. of Engineering & Computer Science
Virginia State University
Petersburg, VA 23806

Acknowledgement

- Many slides are or are revised from the slides of the author of the textbook
  - Matt Bishop, Introduction to Computer Security, Addison-Wesley Professional, October, 2004, ISBN-13: 978-0-321-24774-5. Introduction to Computer Security @ VSU's Safari Book Online subscription
  - http://nob.cs.ucdavis.edu/book/book-intro/slides/

Outline

- Overview
- Access Control Matrix Model
- Protection State Transition
  - Commands
  - Conditional Commands
- Examples
  - Using Access Control Matrix to *model* access control in Linux and Windows

Policy and Mechanism

- Security policy
  - A statement of *what is allowed* and *what is not allowed*
  - Example
    - A student may not copy another student's homework
- Security mechanism
  - A method, tool, or procedure for enforcing security policy
  - Technical and non-technical
    - A homework electronic submission system (e.g., Blackboard) enforces who may read a homework submission

Questions

- Is a given computer system satisfies a given security policy?
- Could we prove a given computer system to be secure?
- Use a generic security policy to determine under what conditions we can prove systems to be secure
  - Access control matrix model
  - Harrison-Ruzzo-Ullman model
  - Take-Grant Protection model
  - A foundation (although “disappointing and incomplete”)

Protection State of System

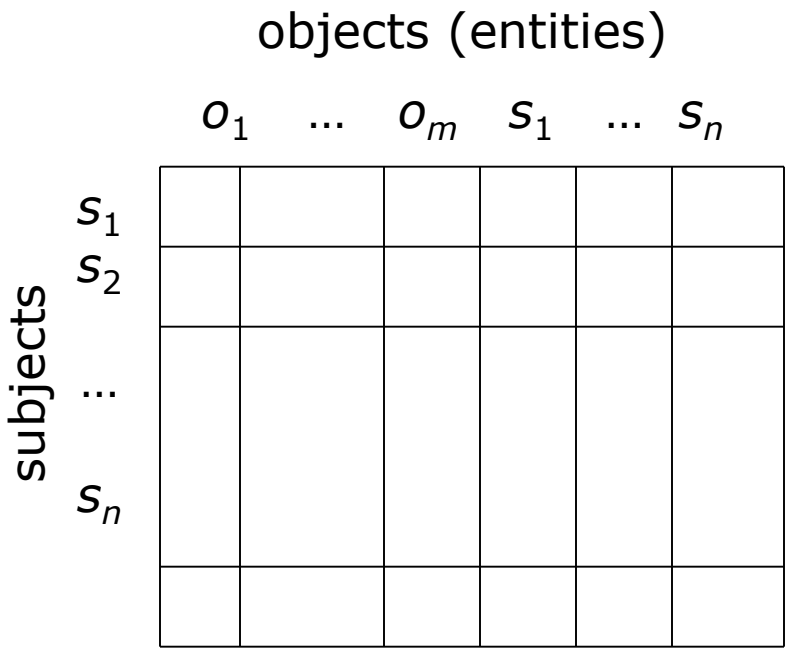
- Protection System
  - Describes the conditions under which a system is secure
- State of System
  - Collection of the current values of all memory locations, all secondary storage, and all registers and other components
- Protection State of System
  - A subset of the state of a system
  - Current settings and values of system relevant to protection

Access Control Matrix

- Represents the current protection state of a system
- Uses a matrix to describe allowed accesses
- Precise model to describe a protection state
  - Specifies the rights of each subject (an active entity, e.g., a process) with respect to every other entity
  - State transitions change elements of the matrix

Access Control Matrix Model

- 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 *p1*, *p2*
- Files *f1*, *f2*
- Rights *r (read)*, *w (write)*, *x (execute)*, *a (append)*, *o (own)*

	f1	f2	p1	p2
p1	*rwo*	*r*	*rwxo*	*w*
p2	*a*	*ro*	*r*	*rwxo*

Example 2

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

	counter	***inc_ctr***	***dec_ctr***	***manage***
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

Linux Access Control: Part I

- Understand Linux file structure
- Linux users and groups

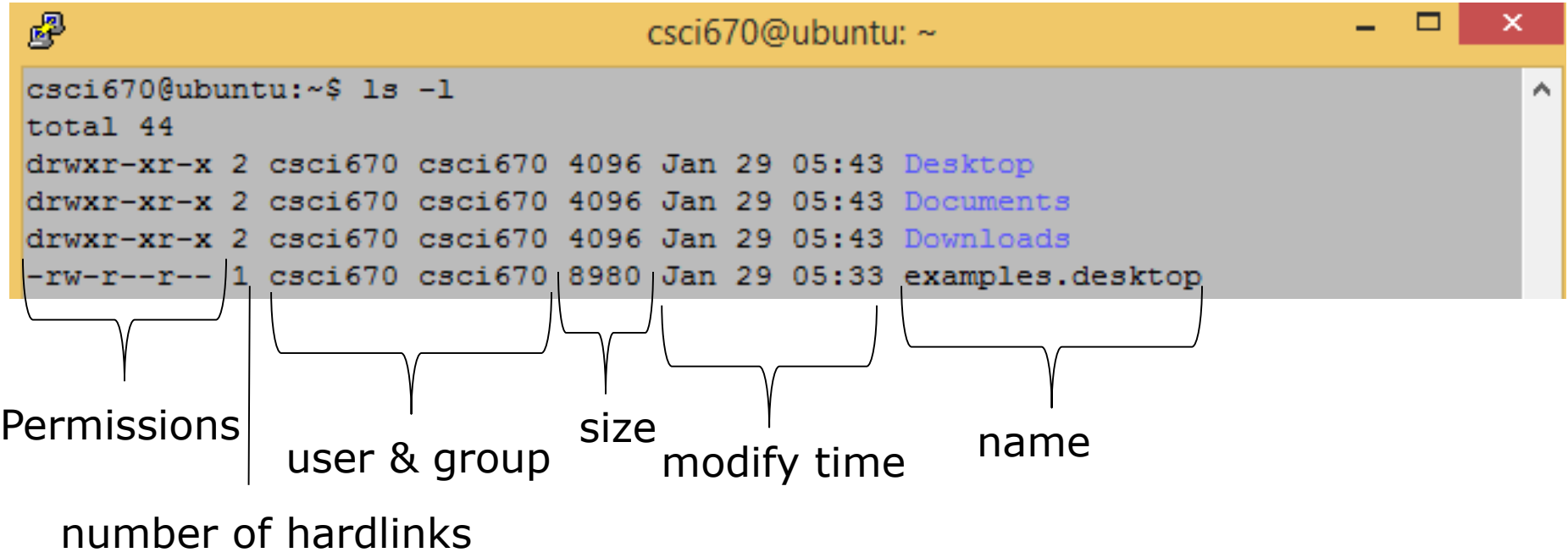
Linux File Structure

- Tree structure
  - Objects: files and directories
  - Directories can have descendants
    - Descendants: files and directories
  - Root of the tree: /
  - Every directory contains two files: • and • •
    - • : current directory
    - • • : parent directory
- Owner and group
  - Every file and directory has an owner and group that it belongs to

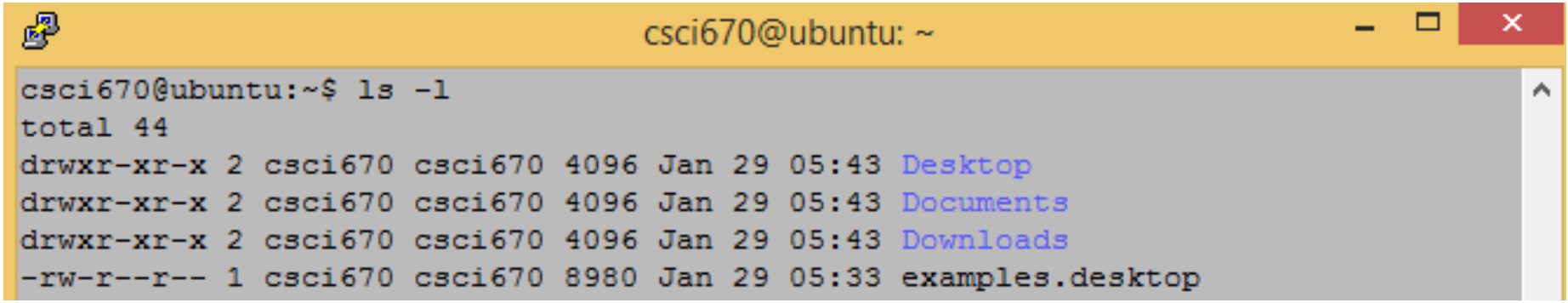
View File and Directory Owner and Group

- Use command ls
  - ls -l

Example of File Listing



File Listing: Explanation



- View Linux info page
  - info ls
    - Then "what information is listed::"
- First letter: object type (d for directory, - for regular file)
- Owner permission, group permission, permission for others (3, 3, and 3 characters, r for readable, w for writable, x for executable, - for permission not granted, s or S for set-user-id or set-group-id bit set)
- Who owns the object (user and group)?

Exercises L2-1

- Question 1(b), 1(d), 1(f) in Exercises 1.11 in the textbook (page 22)

Exercises L2-2

- Describe access rights using an access control matrix for the following files in a Linux system

```
drwxr-xr-x 2 u1 users 4096 Aug 20 09:36 d1
-rw-r--r-- 1 u1 users    4 Aug 20 09:36 f1.txt
-rw----r-- 1 u1 users    4 Aug 20 09:36 f2.txt
-rw----r-- 1 u2 users    4 Aug 20 09:36 f3.txt
```

Homework 1

- Question 1(a), 1(c), 1(e) and 1(g) in Exercises 1.11 in the textbook (page 22)
- Question 1(a) in Exercises 2.6 in the textbook (page 35)

Change Access Rights

- Chang permission (chmod)
  - man chmod
  - Add permission: chmod who+permission filename
  - Remove permission: chmod who-permission filename

```
$ vi hello.cpp
$ g++ hello.cpp -o hello
$ ls -l hello
-rwxr-xr-x 1 hchen users 7867 Feb  5 15:08 hello
$ chmod g+w hello
$ ls -l hello
-rwxrwxr-x 1 hchen users 7867 Feb  5 15:08 hello
$ chmod o-rx hello
$ ls -l hello
-rwxrwx--- 1 hchen users 7867 Feb  5 15:08 hello
```

Change Ownership

- Change file owner and group
  - chown
  - man chown
- Change group ownership
  - chgrp
  - man chgrp

Exercise L2-3

- Document your result in a document
- Create a simple C++ program, such as the Hello, World program

1. Compile the program and run the executable
2. List both source code file and the executable in long format (ls -l)
3. Remove all permission for others from the executable and list the result in long format
4. Add writable permission to the group owner of the executable and list the files in long format
5. Remove executable permission for everyone from the executable, list the result, and make an attempt to run the program (does the program run, why?).

State Transition

- Change the protection state of system
- |– represents transition
  - $X_i \vdash_{\tau} X_{i+1}$: command τ moves system from state X_i to X_{i+1}
  - $X \vdash^{*} Y$: a sequence of commands moves system from state X to Y
- Commands often called *transformation procedures*

6 Primitive Operations

- create subject *s*
  - Creates new row, column in Access Control Matrix (ACM)
- create object *o*
  - creates new column in ACM
- destroy subject *s*
  - Deletes row, column from ACM
- destroy object *o*
  - 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*

Linux Example

- Create File
- Spawn Process
- Make Owner
- Grant Read File Right

Creating File

- Process *p* creates file *f* with owner read (*r*) and write (*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
```

Spawning Process

- Process *p* creates a new process *q with owner read (r) and write (w) permission.*

```
command spawn•process(p, q)
    create subject q;
    enter own into A[p, q];
    enter r into A[p, q];
    enter w into A[p, q];
end
```

Making Owner

- 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

- Execution of some primitives requires that specific preconditions be satisfied.
- Example
  - Let process *p* give process *q* read (*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
```

Testing for Absence of Rights?

- Negation of a condition is not permitted in Access Control Matrix model.
- One cannot test for the absence of a right within a command by the condition,
  - ~~**if r not in A[p, f]**~~
- Further reading
  - Chapter 3
  - Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. 1976. Protection in operating systems. *Commun. ACM* 19, 8 (August 1976), 461-471. DOI=10.1145/360303.360333. http://doi.acm.org/10.1145/360303.360333
  - Sandhu, R.S.; Ganta, S., "On testing for absence of rights in access control models," *Computer Security Foundations Workshop VI, 1993. Proceedings* , vol., no., pp.109,118, 15-17 Jun 1993, doi: 10.1109/CSFW.1993.246635

Exercise L2-4

- Document your result in a document
- In In-Class Exercise L2-3, you are asked to create a simple C++ program, compile and run the program, and change the permission of the executable file. In this exercise, list the operations that results the transition of the protective state using the 6 primitive operations and resulting access control matrix.

Exercise L2-5

- Let us look at question 2(a) and 2(b) of Exercise 2.6 in page 35 of the textbook.

Homework 2

- Question
  - How much memory may be needed to implement the Access Control Matrix model in Linux?
  - Question 1(b) in Exercises 2.6 in the textbook (page 35)
  - Question 2(b) in Exercises 2.6 in the textbook (page 35)

Summary

- Protection state of computer system
- Access control matrix model
  - simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions
- Describe access rights using access control matrix model in Linux and Windows