Access Control

ECE568 – Lecture 17

Courtney Gibson, P.Eng.
University of Toronto ECE

Outline

Access control in operating systems

Access control lists

- Access control matrix

Capabilities

- Problems with capabilities
- The Kerberos system

Mandatory access control

- SELinux
- Role-based access control (RBAC)

O/S Access Control

Access Control Policy

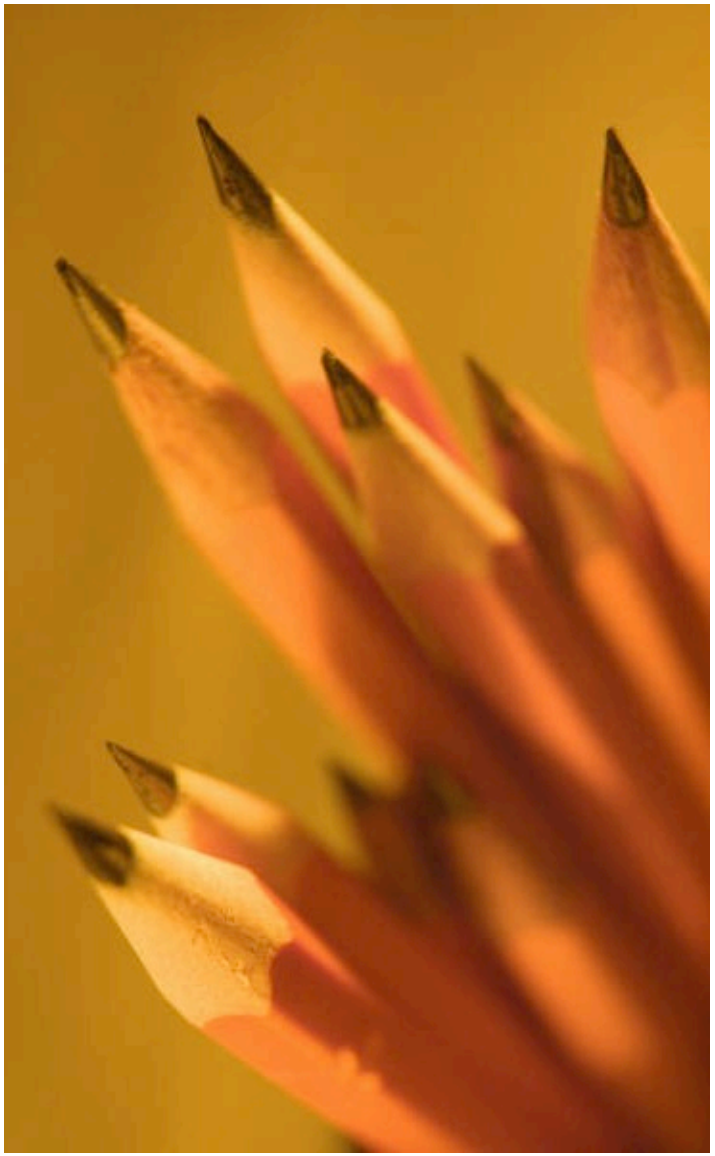
An important function of any security system is to decide whether principals are **authorized** to **access** an object (occurs after user authentication)

- Access control requires two components: a **policy** and an **enforcement mechanism**
- A policy must be specified to indicate the following
  - Who (principal) can access a file (object)?
  - How can they access it (action)?
- An **access control policy** specifies that an **action** performed by a **principal** on an **object** is allowed
  - Actions include, read, write, execute, append, change protection, delete, etc.

Access Control Mechanism

An **enforcement mechanism** is used to implement the policy

- There are two common types of mechanisms
  - **Access Control Lists**
  - **Capabilities**
- These mechanisms are implemented using a **reference monitor**
- A reference monitor has three properties
  - It is **tamperproof**, it is **always invoked** when objects are accessed, it is small enough that it can be **fully analyzed**



Access Control Lists (ACLs)

Access Control Lists (ACLs)

Access Control Lists (ACL) maintain the access control policy **for each resource**

- A list of principles and actions allowed for a resource is maintained for every resource
- Reference monitor checks every access against this list

UNIX file permissions are an example of an ACL

- Each file has a list of principles that can access the file
- Actions include read, write and execute
  - An owner has the ability to modify permissions, and may have read, write or execute ability
  - Actions are specified for a group of users, as well as everyone else
- The policy for each file is the list of users and their allowed actions (*i.e.*, the contents of the ACL)

Access Control Matrix

An **access control matrix** stores the access control lists for several objects together

- There is a row for every user, a column for every object
- Note that a user can also be an object, as the policy might specify which users may communicate with each other and how they might communicate. (*e.g.*, a user may be allowed to remove a user from the system)
- A matrix specifies the ACL policy for an entire system

Principals

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

Dropping Privileges

Sometimes programs need to run with the privileges of different users

- *e.g.*, FTPD needs to write files as different users

By default, programs run with user ID (UID) of the user that invoked the program

- However, a program running with root UID can change its UID by invoking the Unix **setuid** call
- FTPD can use setuid to drop root privileges and become another user

Gaining Privileges

Sometimes a non-root user needs to run a program that requires additional privileges

- *e.g.*, a user invokes the **passwd** program that writes to a protected password file

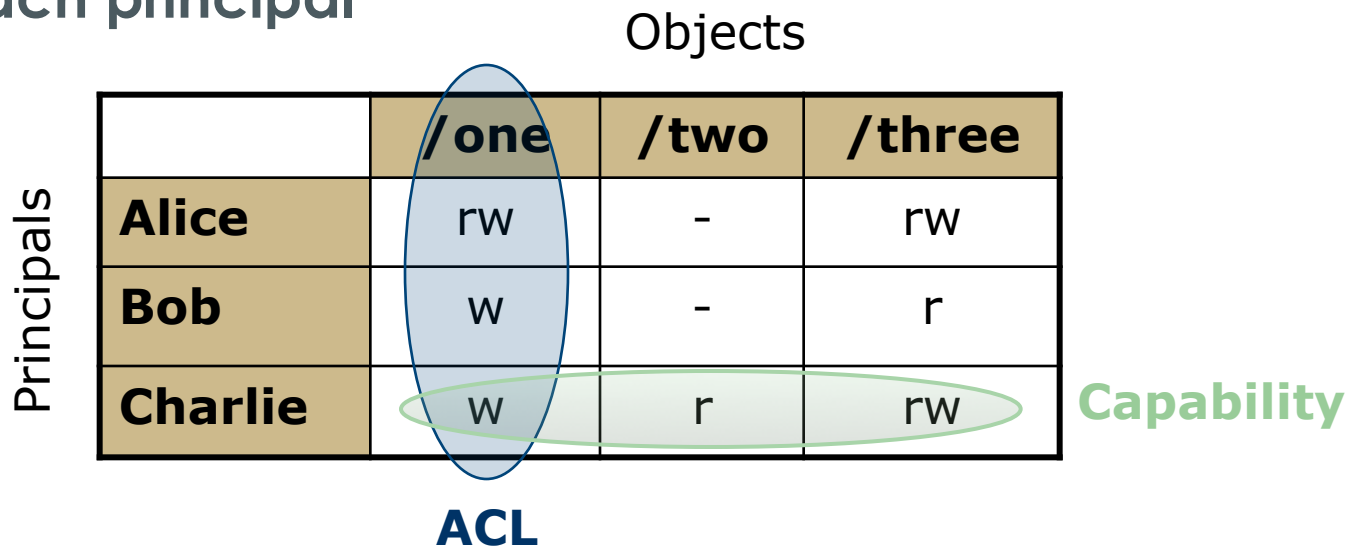
In Unix systems, a user can run a program as another user if the program file has the **setuid bit** turned on

- When a setuid executable is run, it runs with the UID of the owner of the file (rather than the user invoking the program)
- The passwd program file is setuid owned by root and thus runs with root privileges

Capabilities

Capabilities

Capabilities maintain the access control policy **for each principal**



Capabilities are **unforgeable tokens** that a principle presents to the reference monitor to gain access to an object

- Think of a capability as a **key** to a locked resource

Creating & Managing Capabilities

The creation of capabilities must be controlled so that principles cannot arbitrarily **forge** capabilities

- A policy determines how capabilities are created and issued, and typically a central trusted authority (*e.g.*, OS) manages the policy
  - Capabilities can be passed or copied from one principle to another, but the copying of capabilities needs to be controlled
  - The system may restrict the copying or passing of certain capabilities

Implementing Capabilities

Three mechanisms for implementing capabilities:

Tagged Memory

- A tag is associated with every memory location indicating whether a capability is stored there
- Locations tagged as capabilities cannot be modified by processes

Page or segment protection

- A technique similar to tags is to dedicate a region of a process' address space to the storage of capabilities
- You don't need hardware tagging per memory location
- Less memory efficient

Cryptography

- Capabilities are issued and signed by the issuer
- The reference monitor ensures authenticity by checking the signature on each capability, but it can't control copying

"Confused Deputy" Problem

A **Confused Deputy** is a buggy program that is fooled into misusing its privileges:

- Access control lists suffer from this problem:
  - A buggy ftpd program may allow Bob (*e.g.*, the remote client logged in as "Bob") to read and write Alice's files
  - A buggy passwd program may allow a user to get a root shell

Capabilities can solve this problem: each request is accompanied by the capabilities needed to complete the request

- Bob only possesses the capability to access Bob's files

Disadvantage of Capabilities

Capability systems have several disadvantages:

- They generally have poor performance
  - A trap into the kernel is required whenever a capability is manipulated or created
- Revocation of capabilities is difficult
  - Either the capability is found and destroyed (complex if there are several copies), or all services have to be informed to ignore the capability

Capability systems are fairly rare these days

- However, capabilities are used in combination with access control lists
- For example, file descriptors in Unix serve as capabilities
- Can you think of other examples?

Example: The Kerberos System

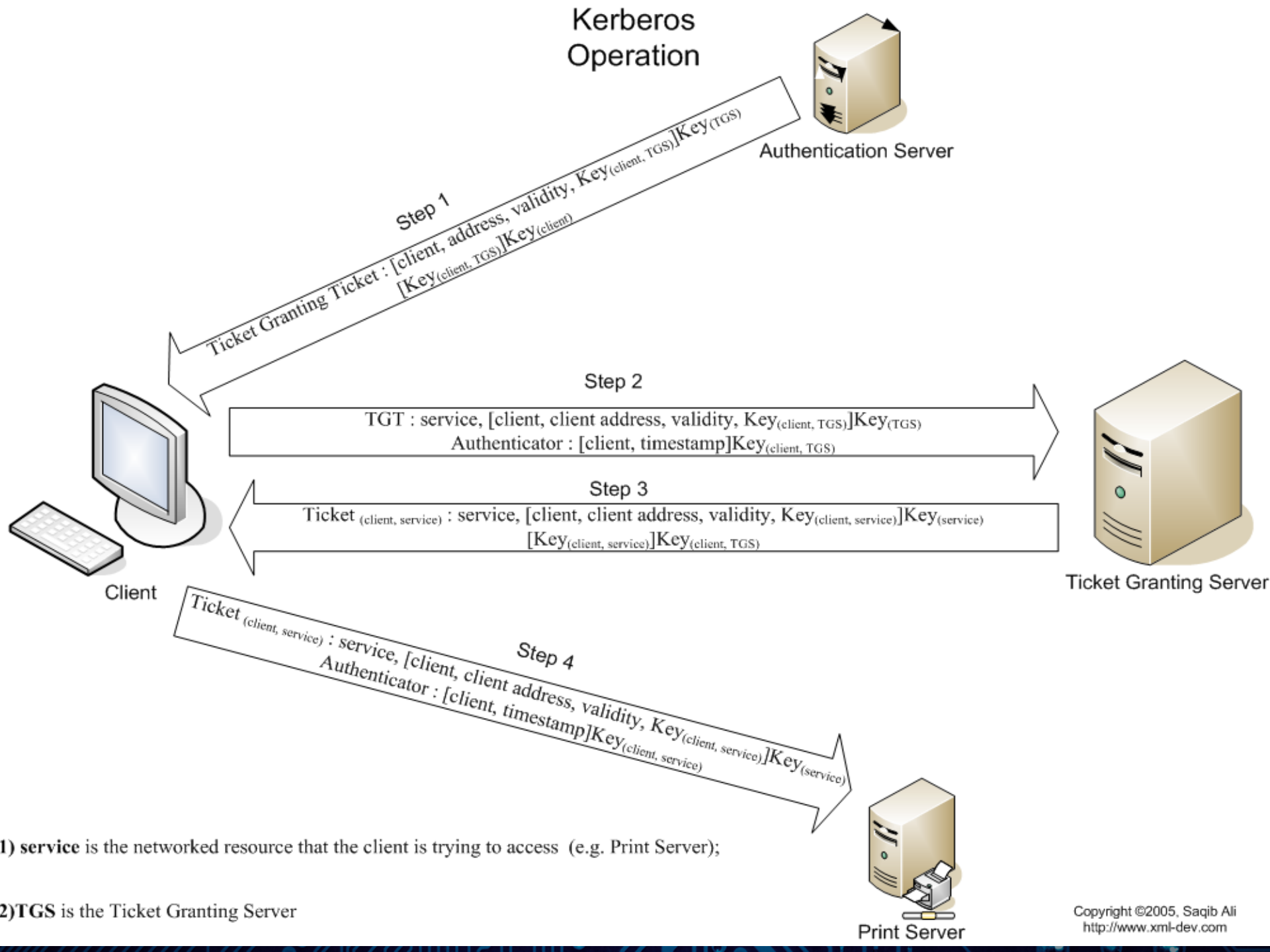
Kerberos is a network wide authentication service that controls access to a set of servers:

- It uses a trusted third party and the symmetric Needham-Schroeder protocol for key exchange

The trusted third party consists of an **authentication server** and a **ticket-granting server**

- The authentication server is used for user authentication
  - Knows all user passwords
  - It provides tickets-granting ticket to a user indicating that the user is authorized to use the ticket-granting service
- The ticket-granting server performs access control
  - It provides **tickets** to a user indicating that the user is authorized to use some service
  - Knows secret keys of all services

Kerberos Operation



1) service is the networked resource that the client is trying to access (e.g. Print Server);

2)TGS is the Ticket Granting Server

Example: The Kerberos System

A ticket is a short-term key that indicates the user's name, what services she can access and an expiry time, and it is encrypted with the secret key of service

- It is essentially a capability that the user presents to gain access to each service
- Windows 2000 and later use Kerberos as their default authentication method
- For more information:
  http://web.mit.edu/kerberos/www/papers.html

Mandatory Access Control

Discretionary vs. Mandatory

There are two types of access control policies:

- **Discretionary Access Control** (DAC)
  - Users are allowed to alter the access control lists on objects that they **own** (*e.g.*, files in a UNIX operating system)
  - Problem: Users can make mistakes with setting up the ACL
- **Mandatory Access Control** (MAC)
  - The system (or administrator) sets up the access control lists and users have no control over changing the list, even for objects they own
  - Example: The OS grants access to some process information (PID, memory usage, etc.), but the process cannot modify this information or share it with others
  - MAC policies generally override DAC policies
  - Problem: Hard to configure

Example: SELinux

A variant of Linux developed by the NSA that enforces mandatory access control

- Released in 2000
- Merged with the main Linux branch (2.6) in 2003

Every object (file, user, program, etc.) has a security label

- Administrator specifies security policies that determine what objects can access other objects based on these labels
  - Can a user execute a program?
  - Can a program access (read, write, modify) a file?
- A program running with elevated (*i.e.*, root) privileges cannot bypass MAC policies even if it is compromised

Example: SELinux

These strict access controls help restrict the application privileges even if the applications are compromised.

Example:

- Web browser cannot access your address book
- A game you download should not read your tax files

Problem is that it is hard to configure

- Have to know all the operations that can be performed by a program, and then create an appropriate policy
- Too strict a policy means the program will fail at times
- Too loose a policy means that if compromised, the program could do damage

Role-Based Access Control

Administration of real systems is very complex

- Example: each file needs permissions for every user
- It would be easier to treat users as groups

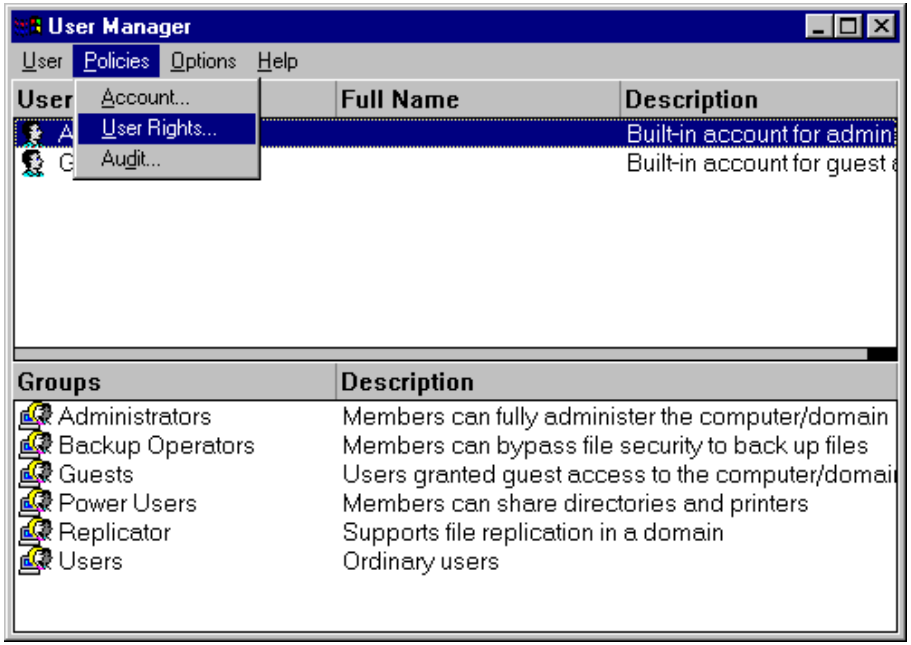
Role-based Access Control (RBAC) defines a set of **roles** with specific privileges, assigned by system

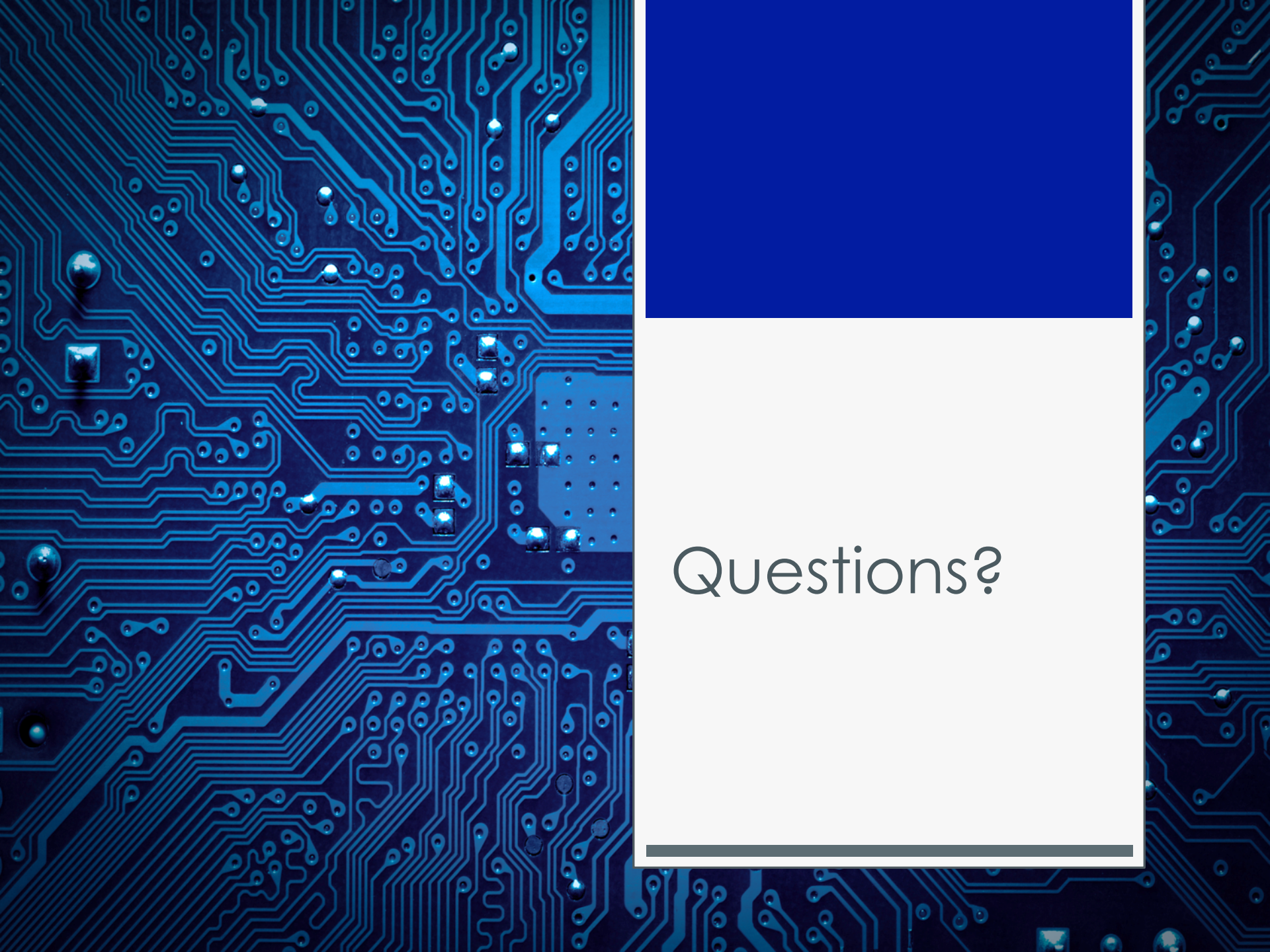
- Users have the ability to take on one or more roles
  - Roles can be things like “developer”, “manager”, or “admin”
  - A user can take on different roles for different tasks
  - Easier for system administrators to think about and assign privileges needed for a role
- RBAC is used in Windows and SELinux
- How are roles different from Unix groups?

Windows Roles

Windows has a couple of built-in roles, and administrators can define more

- When the administrator creates a new user, she assigns the user to one or more roles




Questions?