Recap

- Symmetric crypto system
- Asymmetric crypto system
- Authentication protocols

Authentication Using Public Keys

AP 4.0 uses symmetric keys for authentication

Question: can we use public keys?

Symmetricity in public key crypto: DA(EA(n)) = EA(DA(n))

- DA using private key of A, EA using public key of A

AP 5.0

- A to B: msg = "I am A"
- B to A: once-in-a-lifetime value, n
- A to B: msg2 = DA(n)
- B computes: *if EA(DA(n))== n*
  then A is verified
  else A is fraudulent

Problems with AP 5.0

Bob needs Alice's public key for authentication

- Trudy can impersonate Alice to Bob
  - Trudy to Bob: msg1 = "I am Alice"
  - Bob to Alice: nonce n (Trudy intercepts this message)
  - Trudy to Bob: msg2= DT(n) where DT uses its (Trudy's) private key
  - Bob to Alice: send me your public key (Trudy intercepts)
  - Trudy to Bob: send Trudy's public key (claiming it is Alice's)
  - Bob: verify ET(DT(n)) == n and authenticates Trudy as Alice!!

AP 5.0 is only as "secure" as public key distribution!

PKI – Public Key Infrastructure is to solve the problem. Trusted CA (certificate authority) certificate public keys for others.

Digital Signatures Using Public Keys

Goals of digital signatures:

- Sender cannot repudiate message ("I never sent that")
- Receiver cannot fake a received message

Suppose A wants B to "sign" a message M:

- B sends M and *DB(M)* to A, where DB is decryption with B's private key
- A checks if *EB(DB(M)) == M*, where EB is encryption with B's public key

  If yes, then B has signed M

OS Security

What is the role of Operating Systems in security?

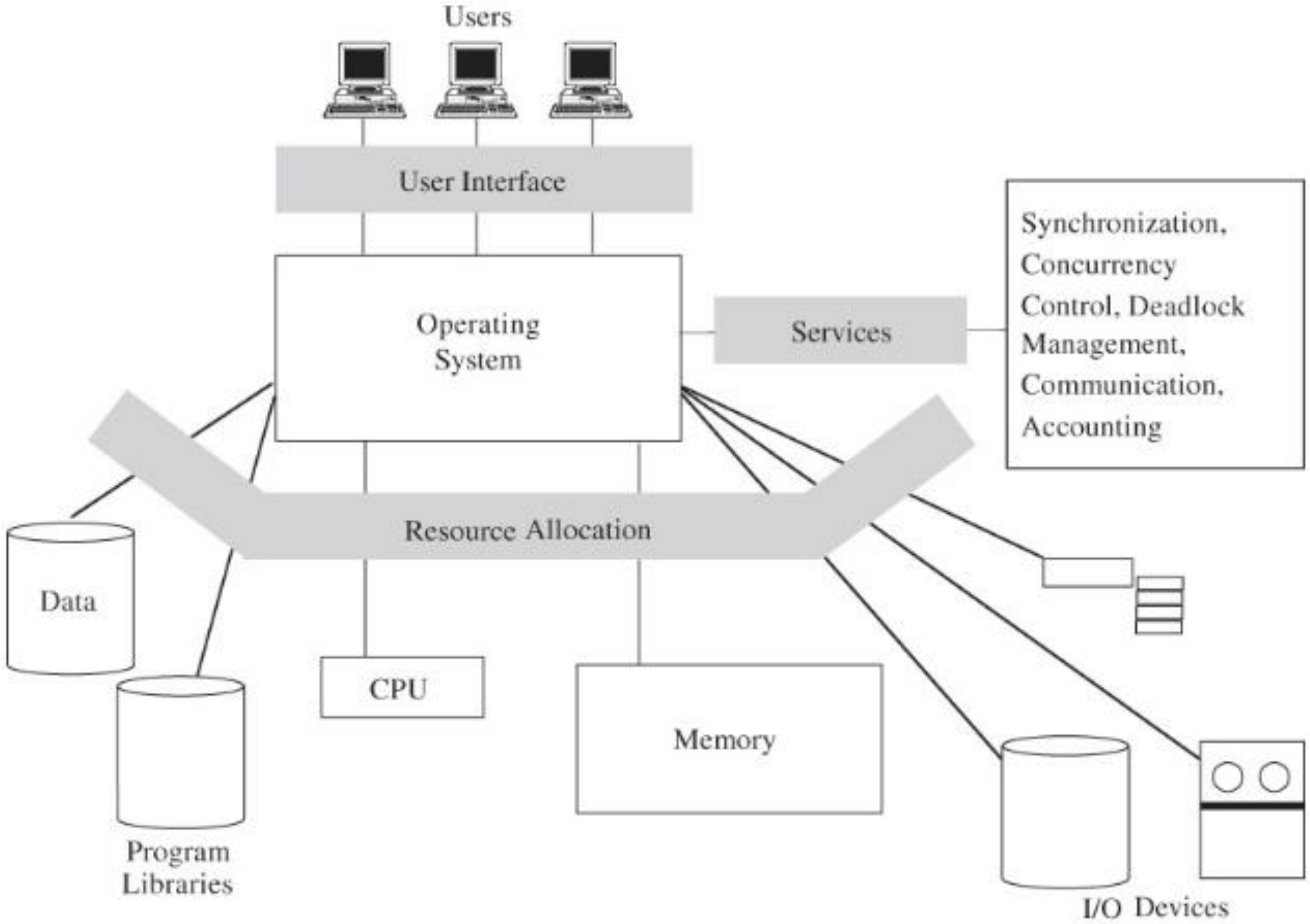
Slides revised from
Matthew Tancreti
Iowa State University

Operating System Viewed as Resource Allocator

Job of OS is to provide access to shared **resources**

OS security is concerned about:

- **Access control** – who should be allowed to access a resource?
- **Integrity** – how to prevent corruption of resources (e.g., data storage and communications)?
- **Availability** – how to ensure resources are available?



Common OS Security Features

Enforced sharing – resources should be shared only as appropriate

- Access control tables is common enforcement mechanism
- Table lookup is performed on every access

Interposes communication and synchronization – OS mediates communication between processes

- Pipes or shared memory need to restrict which processes can read or write
- Ends of a pipe are file descriptors which can only be shared by forking a child process

Protection of critical OS data – the OS must protect its own secret data (e.g., keys) from users

Guaranteed fair service – a process should not be able to “game the system” to get extra resources

- Recall a simple MLFQ allows a process to starve others
- Lottery scheduler and Completely Fire schedule are examples of guaranteeing fairness

Common OS Security Features (cont.)

Interface to hardware – processes need access to hardware resources, the access should be restricted to appropriate use

User authentication – need method to identify users and verify they are who they purport to be

Memory protection – processes should be limited in what memory they can access

- Must prevent processes for accessing each others or the kernels memory inappropriately
- Paging and segmentation enforced by the hardware MMU provide protections

File and I/O – need to protect files from unauthorized users

- Common mechanism is access control matrix (to be discussed more)

Allocation and access control to general objects – other features that need to be protected include concurrency and synchronization

Principles for Secure System Design

Economy of mechanism – keep systems as small and simple as possible, the more complex a system becomes the more likely it is to have vulnerabilities

Fail-safe defaults – default to the secure option over the insecure

Complete mediation – check if action to be performed meets security policy *every single time* action is to be taken, for example, access control

Open design – Assume the adversary knows every detail of the system's design, for example, encryption should not rely on the secrecy of the algorithm, only the key

"The Protection of Information in Computer Systems" by Jerome Saltzer and Michael Schroeder. Proceedings of the IEEE, Vol. 63, No. 9, September 1975. A highly influential paper, particularly their codification of principles for secure system design.

Principles for Secure System Design (cont.)

Separation of privilege – require separate parties or credentials to perform critical actions, for example, two-factor authentication

Least privilege – give user or process the minimum privileges required to perform action

Least common mechanism – for different users or processes, use separate data structures or mechanisms to handle them, for example, each process has its own page table

Acceptability – if a security mechanism is so burdensome that users bypass or don't use it, then it is worthless

"The Protection of Information in Computer Systems" by Jerome Saltzer and Michael Schroeder. Proceedings of the IEEE, Vol. 63, No. 9, September 1975. A highly influential paper, particularly their codification of principles for secure system design.

Defense in Depth

A basic principle of security design is **defense in depth**, multiple layers of security controls (defenses) protect the system

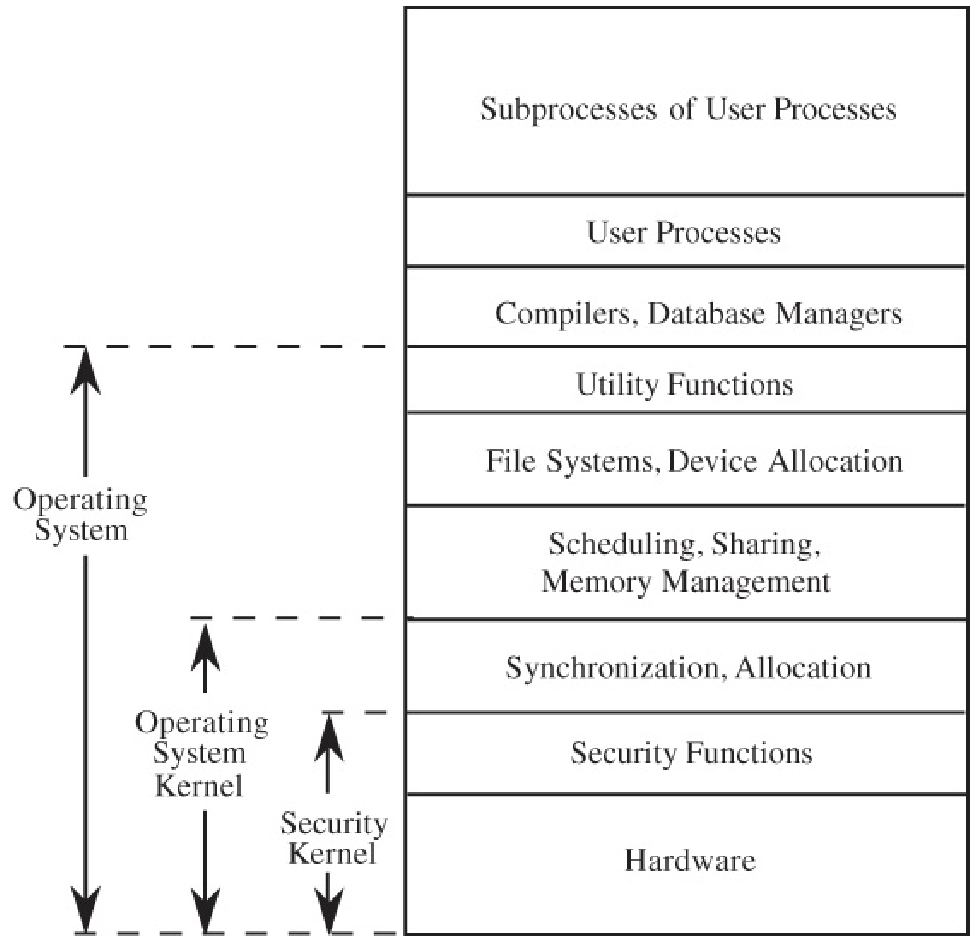
Protect the system using several independent methods, reduces chance that one one vulnerability can compromise entire system

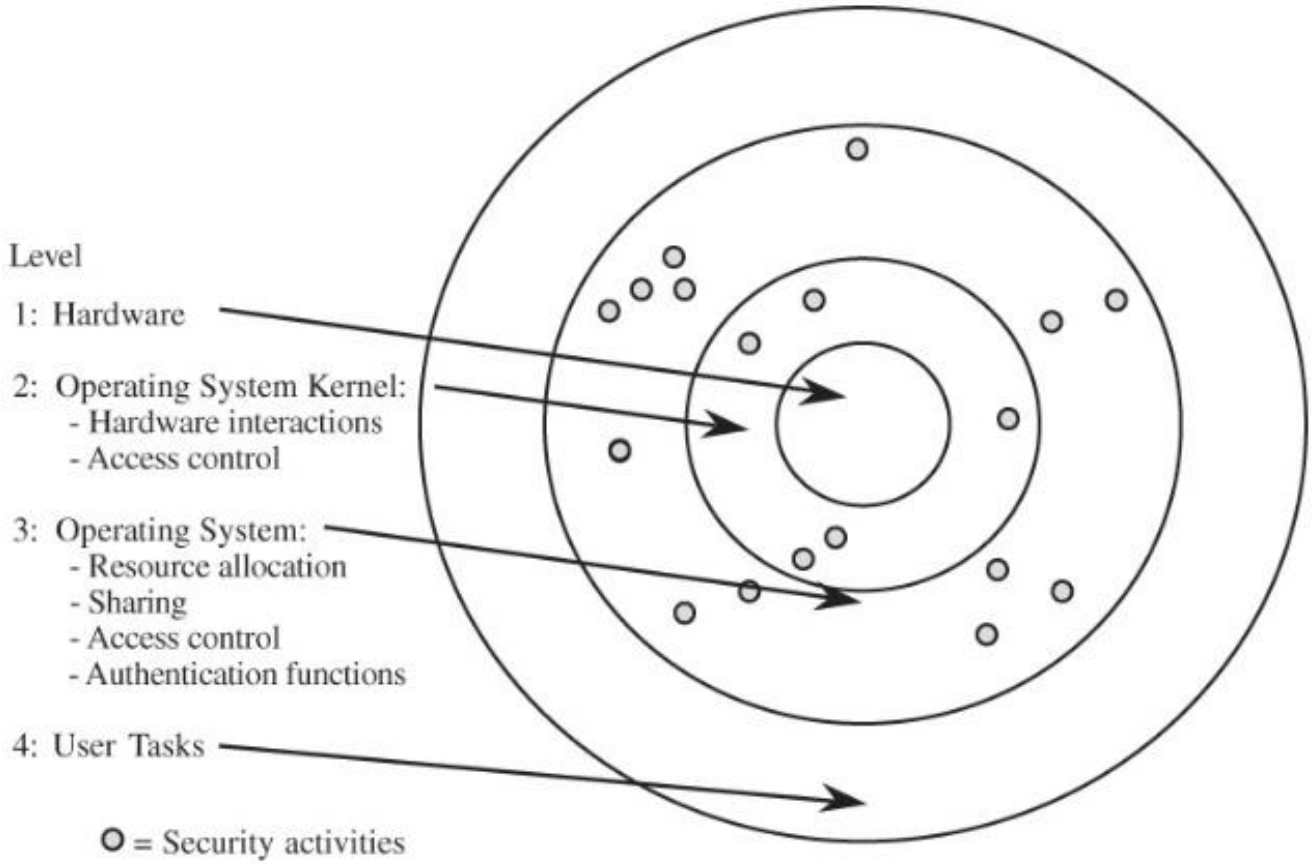
Often conceived of as rings of security, the inner most core of the system is protected by the most layers

Layered Architecture

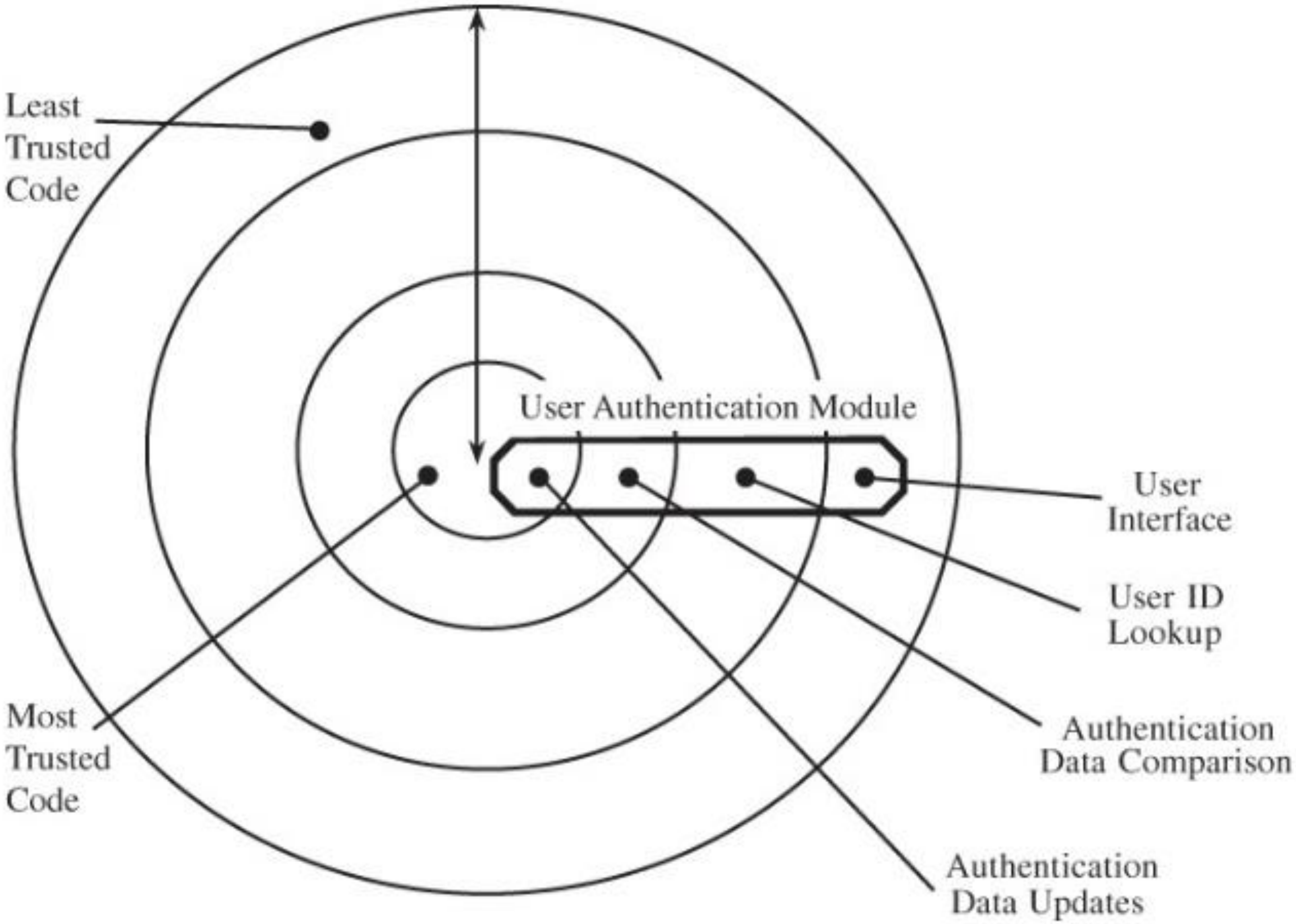
Layered architecture follows the principle of defense in depth

Often relies on hardware support to protect inner layers, e.g., kernel mode and machine mode





Example of Authentication in Layers

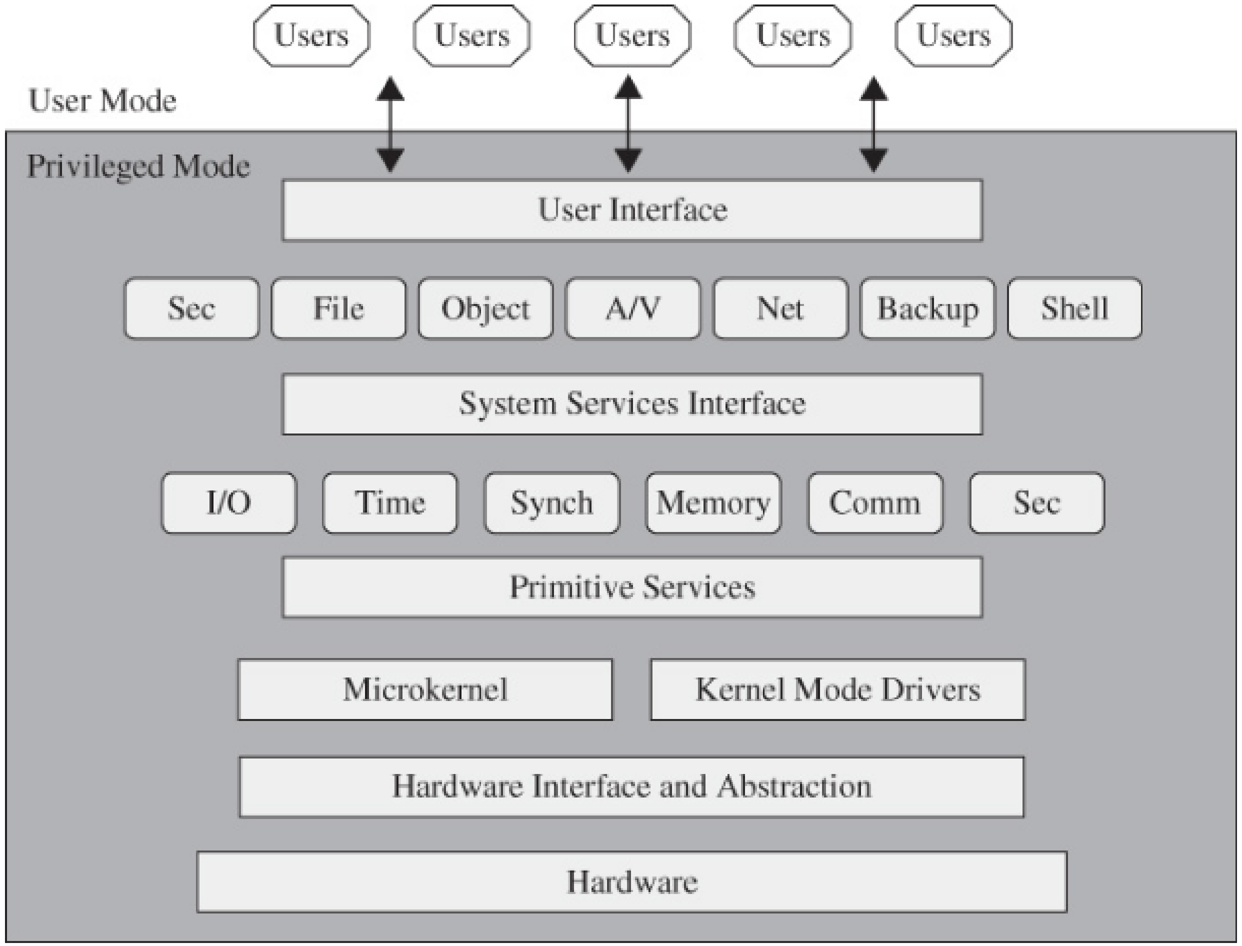


Trust: Motivation for Layered Architecture

A strong motivation for defense in depth in OSes is **trust**

An OS consists of many modules: utilities, drivers, services (e.g., antivirus) and a kernel

The modules are made by different vendors, do not want to trust all at the same level



Design Principle: Layered Trust

A hierarchically designed system has layers of trust

Layers are isolated to limit effects of problems in one layer

Level	Functions	Risk
2	Noncritical functions	Few disasters likely from noncritical software
1	Less critical functions	Some failures possible from less critical functions, but because of separation, impact limited
0	More critical functions	Disasters possible, but unlikely if system simple enough for more critical functions to be analyzed extensively

Trusted Systems

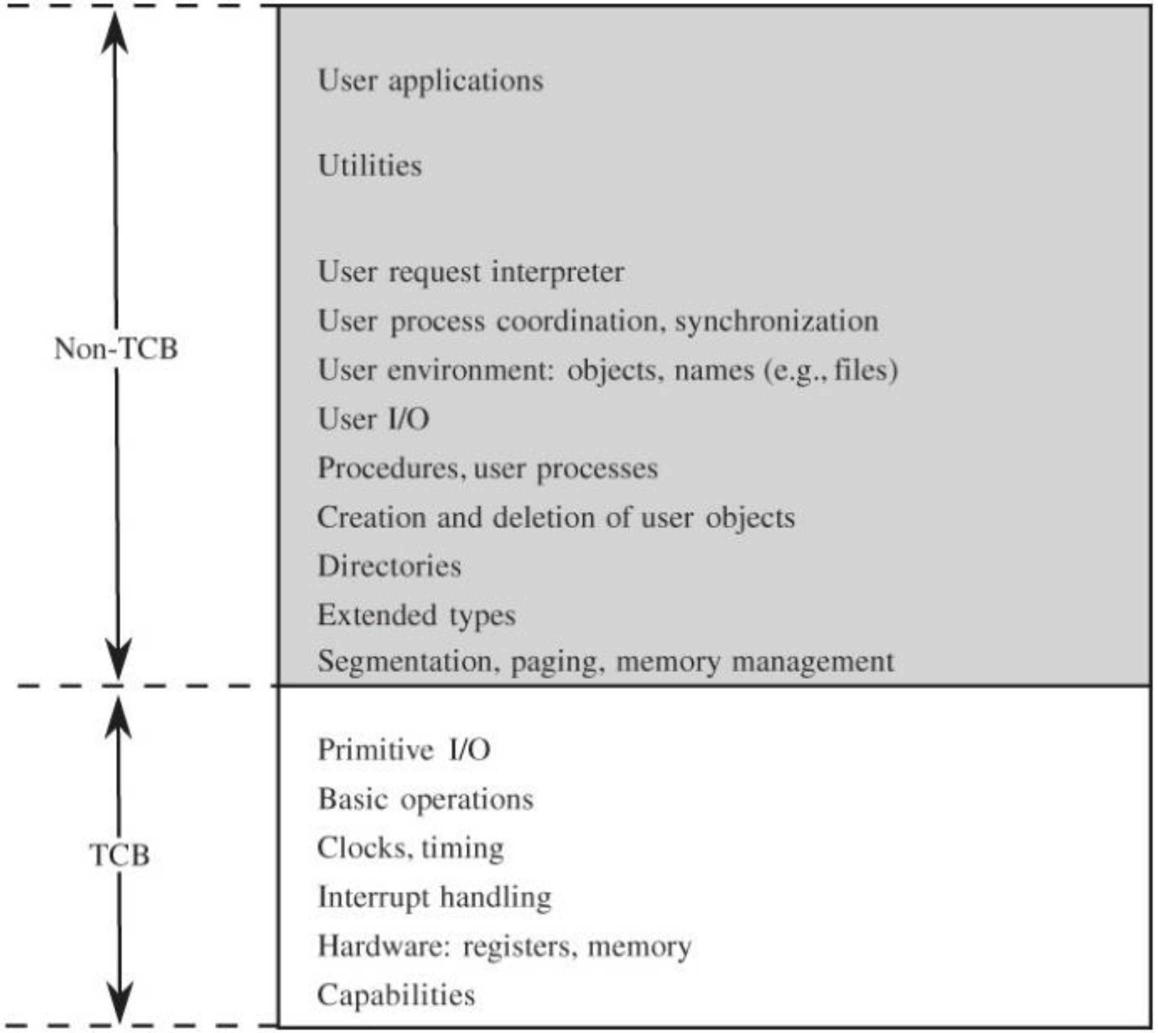
A **trusted system** is one that is relied upon to a specified extent to enforce a specified security policy

Rigorously developed and analyzed, trusted to be tamper-proof, will not execute unauthorized code

Used to act as a kernel or provide critical services (e.g., user authentication, secure boot, digital signature, encryption, etc.)

Trusted Computing Base (TCB)

A **trusted computing base (TCB)** is the name given to everything in the trusted operating system that is necessary to enforce the security policy



Trusted Platform Module (TPM)

A **Trusted Platform Module (TPM)** is hardware that controls what can be done on the machine, for example, it assures booting into intended operating system

Other Operating System Tools for Security

Virtual Machine – present to the users only the resources they need, giving the user the impression their program is running on its own machine

Hypervisor (virtual machine monitor) – software that implements a virtual machine

Sandbox – similar to virtual machine or container, a protected environment in which a program can run and not endanger anything else on the system

Honeypot – a fake environment intended to lure an attacker

Access Control

Access control is regulating what actions subjects can perform on general objects (e.g., files, tables, hardware, network connections, etc.)

Objects are resources on which an action can be performed: files, tables, programs, memory objects, hardware devices, strings, data files, network connection processors, etc.

Subjects human users or agents (e.g., processes) that represent users from which objects are protected

Access mode are controllable actions of subjects on objects, for example: read, write, modify, delete, execute, create, etc.

Traditional Unix File System Permissions

Each file/directory has an owning user and group

Read, write and execute permissions are specified for the user, group and all users

Has many limitations, for example, cannot put file into multiple groups

Symbolic notation	Numeric notation	English
`----------`	0000	no permissions
`-rwx------`	0700	**read, write, & execute only for owner**
`-rwxrwx---`	0770	read, write, & execute for owner and group
`-rwxrwxrwx`	0777	read, write, & execute for owner, group and others
`---x--x--x`	0111	execute
`--w--w--w-`	0222	write

Access Control Matrix

An alternative is the **access control matrix** which describes the **access modes** granted to **subjects** on **objects**

objects

subjects

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

access modes

Access Control Matrix Example

	Bibliog	Temp	F	Help .txt	C_ Comp	Linker	Clock	Printer
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	—	—	R	X	X	R	W
USER S	RW	—	R	R	X	X	R	W
USER T	—	—	R	X	X	X	R	W
SYS MGR	—	—	—	RW	OX	OX	ORW	O
USER SVCS	—	—	—	O	X	X	R	W

Matrix may be sparse (many empty cells)

List of Access Control Triples

A less sparse representation of the access control matrix is a list of access control triples

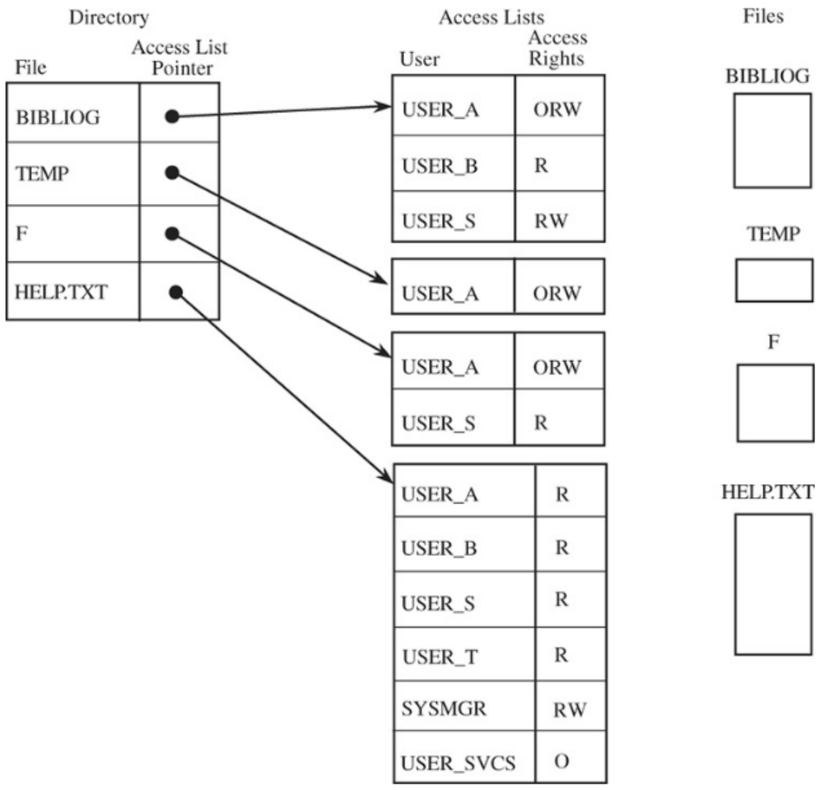
Slow search time, so not often used

Subject	Object	Right
USER A	Bibliog	ORW
USER B	Bibliog	R
USER S	Bibliog	RW
USER A	Temp	ORW
USER A	F	ORW
USER S	F	R
etc.		

Access Control List

An alternative to the access control matrix is the **access control list**

Each column (object) of the access control matrix is stored in a separate list



Access Control Matrix vs List

Access control matrix

- Pro: fast lookup for either subject or object
- Con: matrix is typically sparse, wasted memory

Access Control List

- Pro: Memory efficient
- Con: does not have fast lookup for both subject and object

Example: What algorithm could be used to find all files a user has read access to? Consider both access control matrix and list, which is faster?

Capability

A capability is a tuple of (subject, object, access mode), for example: (User W, Printer, Write)

Capabilities are like tickets some OSes uses to keep track of what processes are allowed to do

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control