OS Security: Access Control

OS Security

- Isolation between user process and kernel process
- Isolation between user processes
- Protection of kernel objects against
  - Unauthorized access
  - Inadvertent modification
- Prevention of unauthorized access
  - Authentication of users
  - Authentication of processes

User Authentication

Authentication Measures

- Something the individual knows
  - password, personal identification number (PIN), secret answers
- Something the individual possesses (token)
  - keys, smart cards, fob, RFID badge
- Something the individual has
  - Physiological biometrics (static): face, fingerprint, iris, retinal scan
  - Behavioural biometrics – speech, handwriting, keyboard dynamics, gait

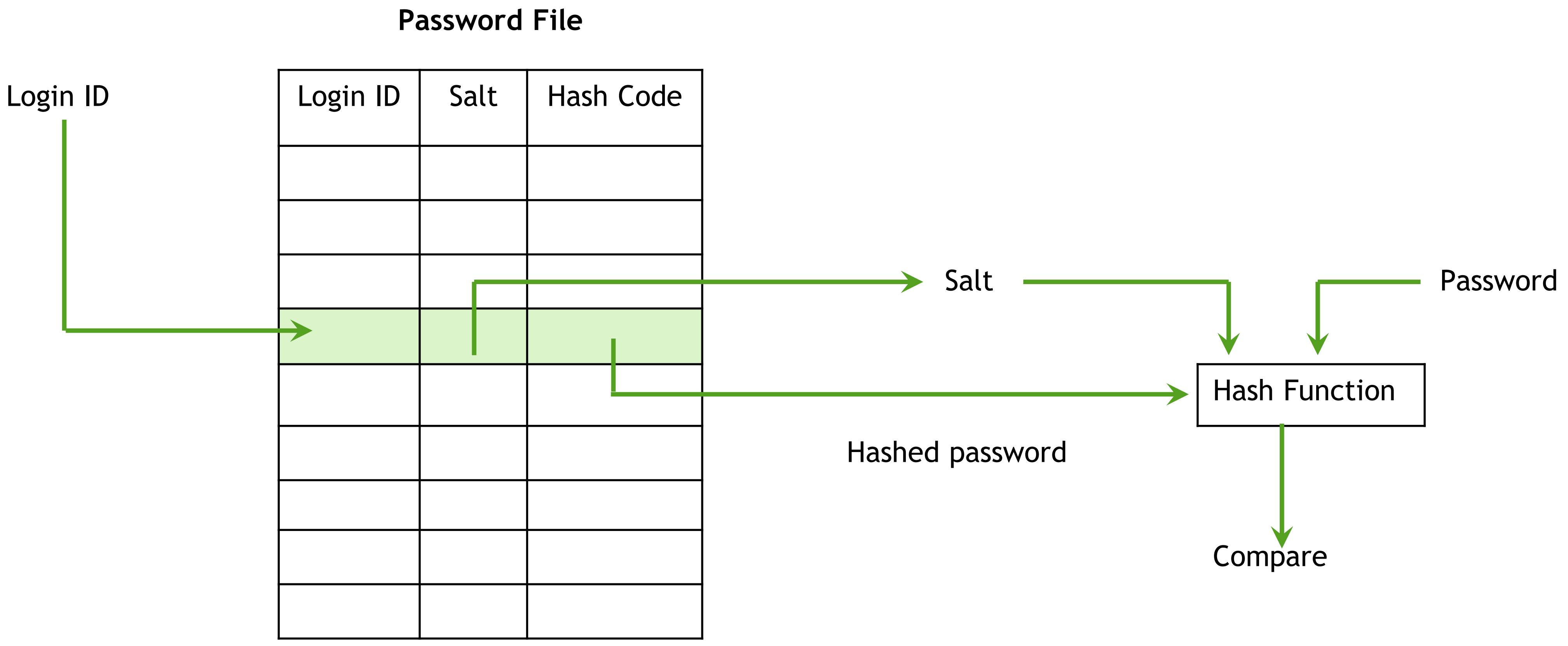
Passwords

- Alphanumeric strings selected by user
- Not stored in cleartext for security reasons
- OS stores hash of the password
- Challenges:
  - User may not select strong password. Most users tend to select passwords that can be memorized easily.
  - Susceptible to dictionary attacks
  - Prone to rainbow attacks: malicious user creates a list of pre-computed hash of passwords

Hardening Passwords

- Enforcing hard passwords
  - Long passwords
  - Password should contain a minimum number of special characters, numerals etc.
  - Change of passwords at specific intervals
- Using salts
  - Salts are pseudorandom sequence of bits, generated by OS
  - Added to password before computing and storing hash
  - Ensures that stored hash is different for two users who happen to use same password
  - User can use same password on different systems
  - Increases complexity of attacks

Password with Salt



Tokens

- Possession based
- Challenges
  - Authenticates whosoever possesses the token
  - Physical keys can be duplicated/lost/damaged
  - Memory card requires special reader and additional security control such as PIN
  - Smart chip card – cryptographic processor; difficult to duplicate

Physiological Biometrics

- Biometric characteristics inherited at birth
- Examples: Face, fingerprint, iris/retinal scans, hand geometry
- Requires input acquisition; algorithmic computations for verification
- Challenges
  - Technical deployment; Complexity
  - Access Time
  - User acceptability
  - Privacy issues
  - False positives; False negatives

Behavioural Biometrics

- Biometric characteristics acquired
- Examples: Speech, handwriting, gait, keyboard dynamics (typing rythym)
- Pattern may change
  - affected by emotional state
  - sickness

Two Factor Authentication

- Two layers of security: User needs to pass security checks for both layers before being granted access
- For additional security, two different channels used for these security checks
- Used in payment gateway authentication
  - Password and (PIN sent on registered email or through SMS on registered mobile)
  - Password and specific numbers corresponding to characters on grid printed on debit card

Access Control

Access Control

- A security policy that specifies what type of access allowed to a subject on a given object and under what conditions
- Implementation requires creation of an authorization database
  - Recording type of access allowed to an user for resource(s)
  - the access control function verifies against this database before granting access
- A log maintained of all user accesses to resources (checked for unauthorized access and identification of bugs in policy and/or implementation)

Security Policy

- Principle of Least Privilege
- Principle of Attenuation of Privilege: a process can never increase its rights, or transfer rights it does not have.

Terminology

- Subjects: An entity capable of accessing objects.
  - user or application actually gains access
- Objects: Any resource to which access is required by user/process
  - Applications, firewall, routers, file, databases, disk blocks, memory segments, software objects
- Access right: type of access allowed to a subject on an object
  - read, write, execute, functions in software objects.
- Domains

Types

- Mandatory Access Control (MAC)
  - Access granted on comparing resource’s security label with subject’s security clearance
  - A subject with requisite clearance can’t grant that clearance to another subject
  - Used for sensitive applications such as military
- Discretionary Access Control (DAC)
  - Access granted on subject’s identity and authorization rules
  - Access can be passed to another subject
- Role based Access Control (RBAC):
  - Access granted as per authorization rules for a role assigned to a subject at that time
  - Effective implementation of the principle of least privilege

Access Rights

- Processes: delete a process, stop (block), and wake up a process.
- Devices: read/write the device, control its operation (e.g., a disk seek), block/unblock the device for use.
- Memory locations or regions: read/write certain locations of regions of memory that are protected, read/write access not allowed.
- Subjects: grant or delete access rights of that subject to other objects, as explained subsequently.

Access Control Matrix

Access Control Matrix

- Rows represent users
- Columns present objects
- Each cell records access rights of the respective subject on the corresponding object

	File 1	File 2	File 3	Directory1
User 1	Own R W	-	R	
User 2	R	Own R W	R	Own R X
User 3	-	-	Own R W	R
User 4	R W	R		R

Access Control Matrix

- A is access control matrix
- A[s,o] is an entry for subject s and object o
- Primitive Operations
  - Create new subject (adding new row)
  - Create new object (adding new column)
  - Destroy subject s (delete a row)
  - Destroy object s (delete a column)
  - Enter right r into A[s,o]
  - Delete right r from A[s,o]
- Monitor: A hardware/software mechanism that controls checks access for an object o for subject s as per A[s,o]]

Primitive Operations

- A’ is modified Access Control Matrix

Operation	Conditions	New State	Description
Create subject s’	s’ ∉ O	S’ = S ∪ {s’}; O’ = O ∪ {s’} A’[s’,o] = ∅, o ∈ O’ A’[s, s’] = ∅, s ∈ S’	Add row for s’ Execute create object s’ i.e. add column for s’ A[s’, s’] = control
Create object o’	o’ ∉ O	S’ = S; O’ = O ∪ {o’} A’[s,o] = A[s,o]; s ∈ S; o ∈ O A’[s,o’] = ∅, s ∈ S’	Add column for o’ Add owner in A[s,o’]
Destroy subject s’	s’ ∈ S	S’ = S - {s’}; O’ = O - {s’} A’[s,o] = A[s,o]; s ∈ S’; o ∈ O’	Destroy object s’ Delete row s’
Destroy object o’	o’ ∈ O o’ ∉ S	S’ = S - {s’}; O’ = O - {o’} A’[s,o] = A[s,o]; s ∈ S’; o ∈ O’	Delete column o’
Enter r into A[s,o]	s ∈ S o ∈ O	S’ = S; O’ = O A’[s,o] = A[s,o] ∪ {r} $A'[s_1,o_1] = A[s_1,o_1] \quad (s_1, o_1) \boxed{\neq} (s,o)$	
Delete r from A[s,o]	s ∈ S o ∈ O	S’ = S; O’ = O A’[s,o] = A[s,o] - {r} $A'[s_1,o_1] = A[s_1,o_1] \quad (s_1, o_1) \boxed{\neq} (s,o)$	

Create Object (File)

```
// Process p creates file f
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
```

Confer Read Rights to Another Subject

```
// Process p confers read rights on file f to process q
Command confer.read(p, q, f)
    If own in A[p,f] then
        Enter R into A[q, f]
    endif
end
```

Revoke Read Rights from Another Subject

```
// Process p revokes read rights on file f from process q
Command confer.read(p, q, f)
    If own in A[p,f] then
        Delete R from A[q, f]
    endif
End
```

Owner of an object can grant a right to the object it does not have. It can grant this right to itself. This allows it to revoke its W-access to an object, and later restore the right to modify the object.

Transfer Read Rights to Another Subject

```
// Process p confers read rights on file f to process q
// R* means read rights with copy allowed
//Q is granted read rights but not allowed to transfer these to any other process
Command confer.read(p, q, f)
        If R* in A[p,f] then
                Enter R into A[q, f]
        endif
End
```

Transfer-only Read Rights to Another Subject

```
// Process p confers read rights on file f to process q
// R+ means transfer of read rights allowed
Command confer.read(p, q, f)
    If R+ in A[p,f] then
        Delete R+ from A[p, f]
        Enter R+ into A[q, f]
    endif
End
```

Control Access Rights of Subordinate

```
// control right needed
Command create.subordinate(p, q, m)
        Create subject q
        Create object m
        Enter control into A[p, q]
        Enter R into A[q, m]
        Enter W into A[q, m]
        Enter E into A[q, m]
End
```

Take/Revoke Access Rights of Subordinate

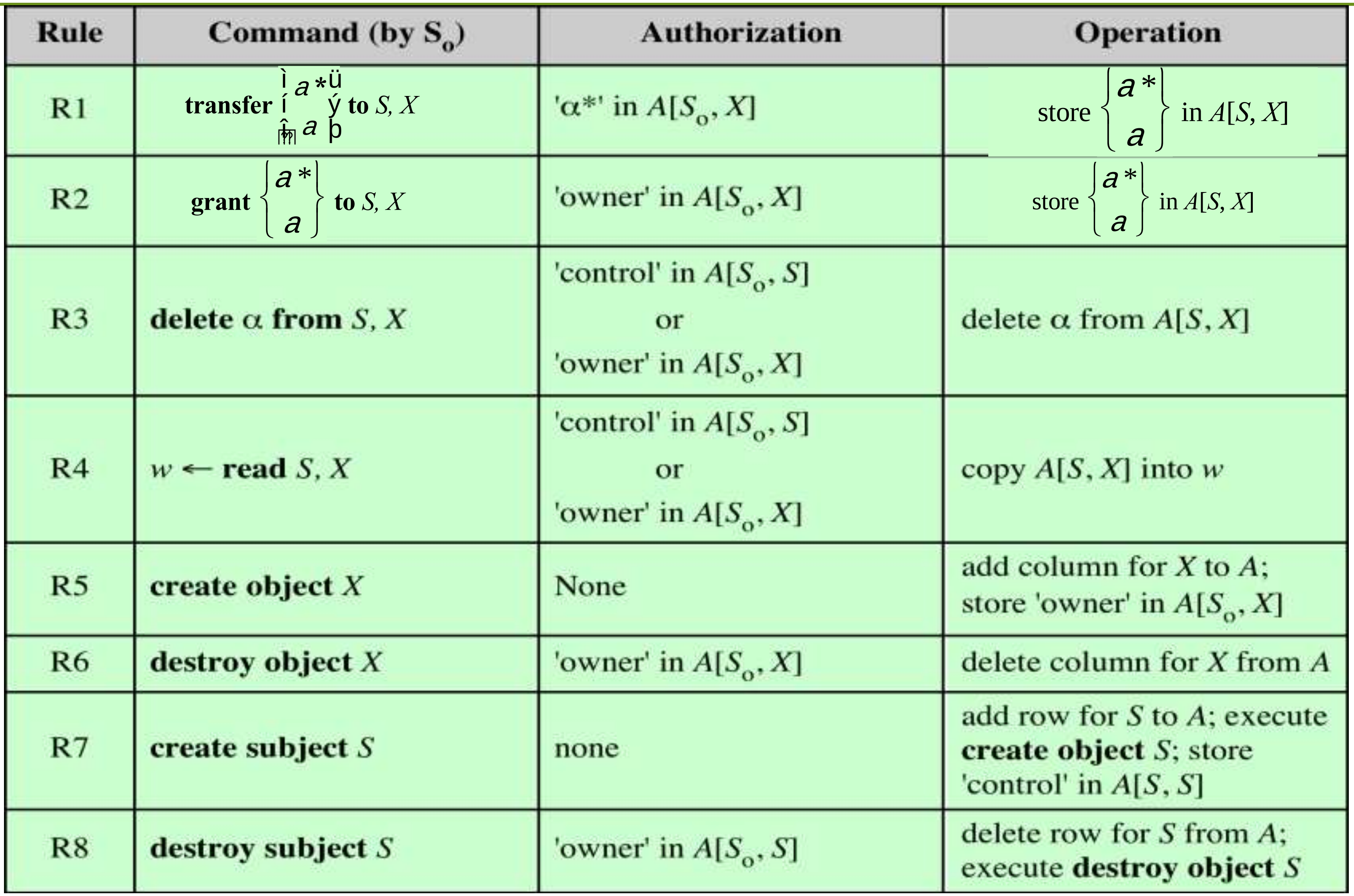
```
Command take.subordinate.read(p, q, m)
    If control in A[p,q] and R in A[q,m]
        Enter R in A[p, m]
    Endif
End

Command revoke.subordinate.read(p, q, m)
    If control in A[p,q]
        Delete R in A[q, m]
    Endif
End
```

Revoke revised

```
// control right needed
Command revoke.read(p, q, f)
    If own in A[p, f] or Control in A[p,q]
        delete R from A[q, f]
    Endif
End
```

Summary



Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'α^*' in $A[S_o, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R2	**grant** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'owner' in $A[S_o, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_o, X]$
R6	**destroy object** X	'owner' in $A[S_o, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_o, S]$	delete row for S from A; execute **destroy object** S

Protection Mechanism

Access Control Mechanism

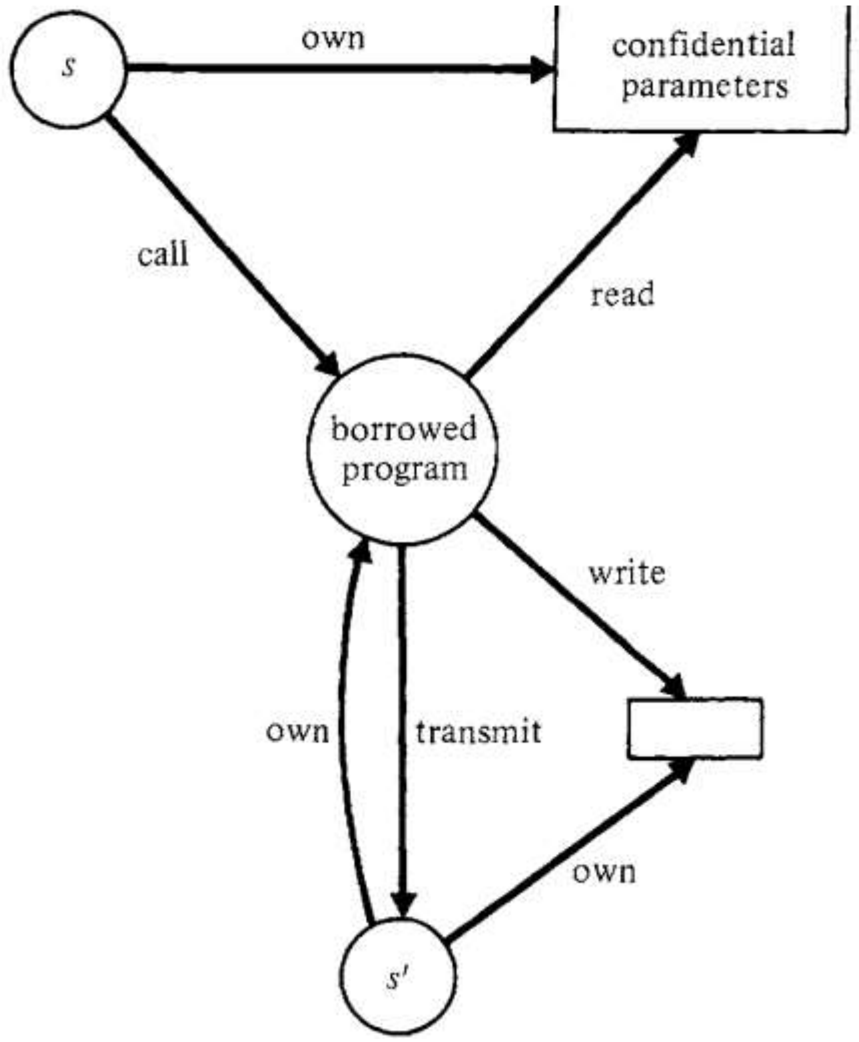
- S = set of all possible states
- P = set of states authorized by the protection
- R = set of states reachable with the security mechanisms
- P and R are subsets of S
- Secure: R ⊂ P; (all reachable states are authorized)
- Precise: R = P (all authorized states are reachable)
- Overprotective: Secure but not precise
- Examples insecure systems
  - read file directly from disk (bypassing file system)
  - System not clear memory between use may expose data to unauthorized subjects.

Sharing

- No sharing (isolation)
- Sharing copies of data object
  - Can the subject be trusted
  - What if subject leaks data to others?
- Sharing originals of data object
  - Time and space saving approach
  - Consistent view
  - Malicious modification of data?
- Sharing of programs: one program calls/invokes another
  - share rights of execution/calling environment
  - Can program be trusted? Trojan horse?
  - Copies/leaks parameters?
  - Execute the program with no ability to read or copy these.

Confinement Problem

- User connects to server for some service
- Needs to pass information to server
- server may leak information deemed confidential by the user



Principles of Protection Mechanism

- Least Privilege: Grant necessary rights at a given time; execute processes in small protection domains
- Economy of mechanism: simple so that can be verified
- Complete Mediation: Every access should be checked for authorization
- Open Design
- Separation of Privilege: Access to object should depend on more than one condition being satisfied
- Least Common Mechanism: Sharing should be minimal
- Acceptability

Access Matrix Implementation

Access Control Matrix

- ❑ Rows represent users
- ❑ Columns present objects
- ❑ Each cell records access rights of the respective subject on the corresponding object

	File 1	File 2	File 3	Directory1
User 1	Own R W	-	R	
User 2	R	Own R W	R	Own R X
User 3	-	-	Own R W	R
User 4	R W	R		R

Access Control Matrix (contd).

- Large matrix that is Sparse. Implemented as
  - Access Control List:
    - Each non-null entry in column stored as a linked list
    - For each object, keeps information on objects and their access rights.
  - Capability List:
    - Each non-null entry in row stored as a capability list
    - For each subject, keeps information on what access is permissible for which objects
- Most systems keep both lists

Access Control Matrix

	File 1	File 2	File 3	Directory1
User 1	Own R W	-	R	
User 2	R	Own R W	R	Own R X
User 3	-	-	Own R W	R
User 4	R W	R		R

File 1	File 2	File 3	Directory1
User 1 Own R W	User 2 Own R W	User 3 Own R W	User 2 Own R X
User 2 R	User 4 R	User 1 R	User 3 R
User 4 R		User 2 R	User 4 R

Access Control List

User 1	File 1 O R W	File 3 R		
User 2	File 1 R	File 2 O R W	File 3 R	Directory 1 O R X
User 3	File 3 O R W	Dir 1 R		
User 4	File 1 R W	File 2 R	Dir 1 R	

Capability List

- Access control mechanisms based on three concepts:
  - Access Hierarchies, which automatically give privileged subjects a superset of the rights of less privileged subjects.
  - Authorization Lists, which are lists of subjects having access rights to some particular object.
  - Capabilities, which are like "tickets" for objects; possession of a capability unconditionally authorizes the holder access to the object.

Access Hierarchy

- Rings : Different levels of access allowed
- Ring 0 has highest privileges
- Most systems support two levels
  - User mode and Kernel/Supervisory mode (privileged mode)
- Privileged process:: can create and destroy objects, initiate and terminate processes, access restricted regions of memory containing system tables, and execute privileged instructions
- Supervisor states and ring structures are contrary to the principle of least privilege. Systems programs typically run with considerably more privilege than required.

Authorization List (ACL)

Authorization List (ACL)

- An authorization list (also called an access-control list) is a list of n >= 0 subjects authorized to access some particular object x.
- For object x, the kth entry in the list gives the name of a subject S_k and the rights R_k in $A[S_k, x]$ of the access matrix
- An authorization list, therefore, represents the nonempty entries in column x of the access matrix.

Sno	Subject	Rights
1	S1	O R W
2	S2	R
3	S3	W
4	S4	R

ACL (contd.)

- ACL implemented as two entries: first with access rights of owner and second with access rights for others.
- Access rights are usually limited to R and W.
- Unix employs three entries: owner; group; others
- Does not meet the objective of least privilege
- Search is expensive; every access not verified by OS
  - OS checks authorization list when a file is opened, but not for each read or write. If a right is revoked after a file is opened, the revocation takes effect only after the file is closed.
  - Not suitable for protecting segments of memory, where address bounds must be checked for every reference.

Capabilities

- A capability is a pair (x, r) specifying the unique name (logical address) of an object x and a set of access rights r for x (some capabilities also specify an object’s type).
- The capability is a ticket that unconditionally authorizes the holder r-access to x.
- Once the capability is granted, no further validation of access is required. Without the capability mechanism, validation would be required on each access by searching an authorization list

Reading Assignments

- Confused Deputy Problem
- Clickjacking
- TOCTOU (Time of Check to Time of Use) Race Condition

OS Security

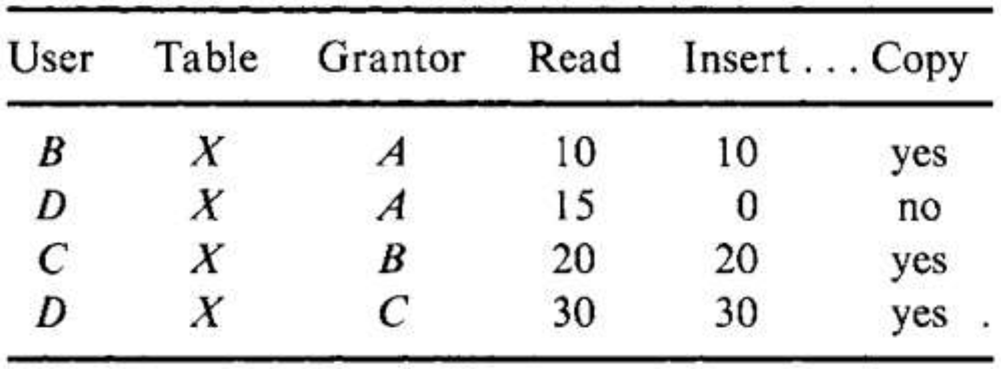
- Isolation between user process and kernel process
- Isolation between user processes
- Protection of kernel objects against
  - Unauthorized access
  - Inadvertent modification
- Prevention of unauthorized access
  - Authentication of users
  - Authentication of processes

ACL : Grant and Revocation

- In most systems, owner (creator) of an object can grant rights on the object to other subjects
- In some systems, acquired right even on non-owned object can be granted to other users
- A is owner of O1. A grants R rights on O1 to B.
- B has R rights but is not owner
- If B is allowed to grant rights to another subject C, what security policy be adopted?
  - B can only transfer rights that it has. It can't transfer W rights.
  - If rights of B are revoked, rights of any subject s on O1 should be revoked only if these rights were granted through B.
  - Example: B is granted R rights; grants these to C which in turn grants these to D. So when B's rights are revoked, C and D R right should also be revoked.

Grant/Revocation of Rights

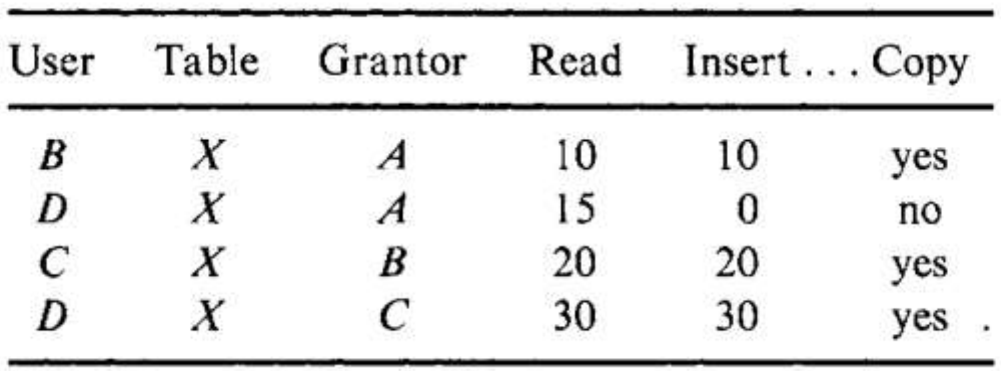
- OS needs to keep information on type of rights granted to a subject on an object. For proper implementation of revocation, it needs to keep information on grantor of rights; if grantor is owner and time when rights were granted.
- Table
- Graph



User	Table	Grantor	Read	Insert	. . . Copy
B	*X*	*A*	10	10	yes
D	*X*	*A*	15	0	no
C	*X*	*B*	20	20	yes
D	*X*	*C*	30	30	yes .

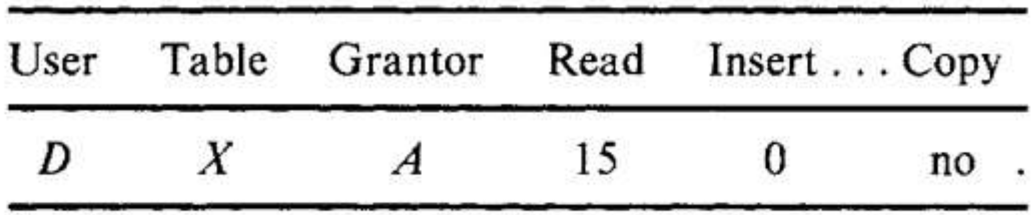
$$\begin{tikzcd}
A \arrow[r, "10", "{R, I}"'] \arrow[rrr, bend right, "15", "R"'] & B \arrow[r, "20", "{R, I}"'] & C \arrow[r, "30", "{R, I}"'] & D
\end{tikzcd}$$

Grant/Revocation of Rights



User	Table	Grantor	Read	Insert . . .	Copy
B	*X*	*A*	10	10	yes
D	*X*	*A*	15	0	no
C	*X*	*B*	20	20	yes
D	*X*	*C*	30	30	yes .

- A revokes rights of B at t =40.
  - B loses rights (1st row is deleted)
  - C loses rights granted by B (3rd row is deleted)
  - D loses rights granted by C(4th row is deleted)

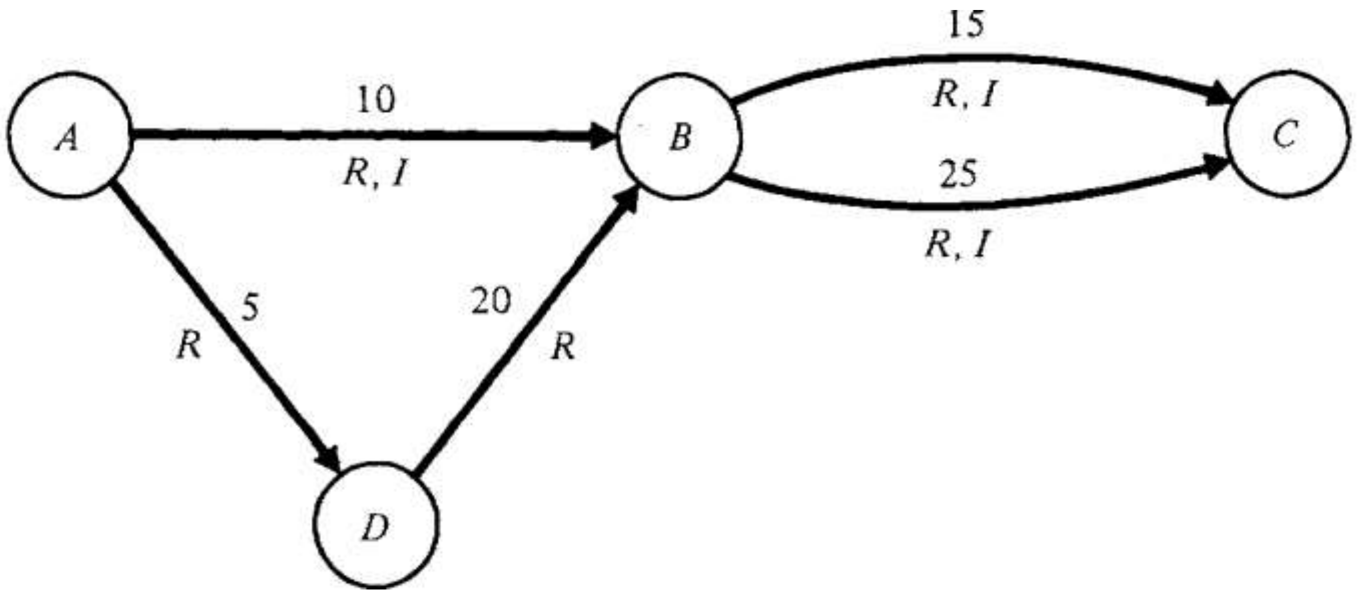


User	Table	Grantor	Read	Insert . . .	Copy
D	*X*	*A*	15	0	no .

Second row is not deleted as D, here, has been granted rights by A.

Grant/Revocation of Rights

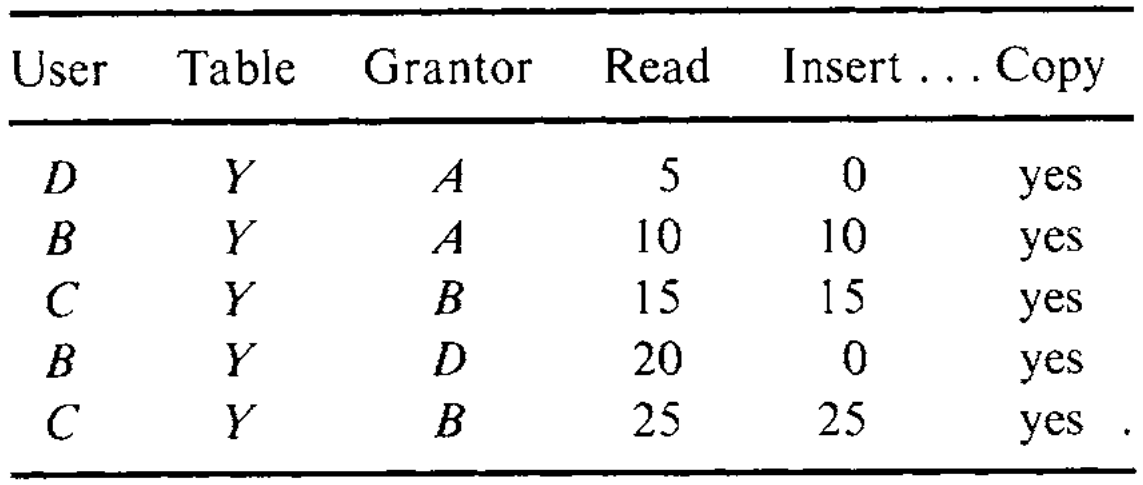
- A grants rights R (read), I (insert) to B at t=10, B grants these rights to C at t=15. D is granted R right by A at t = 5 and grants these rights to B at t = 20. At t=25, B grants R and I rights to C.



- What if A revokes B’s rights at t=40

Grant/Revocation of Rights

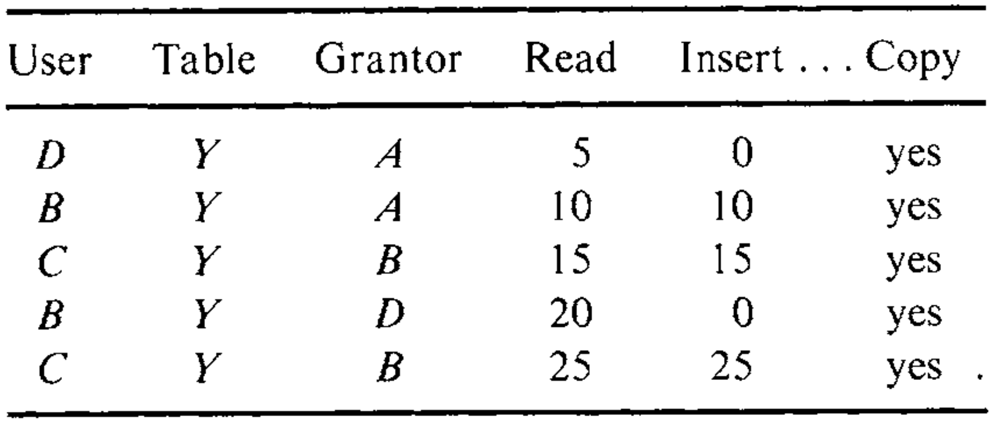
- A grants rights R (read), I (insert) to B at t=10, B grants these rights to C at t=15. D is granted R right by A at t = 5 and grants these rights to B at t = 20. At t=25, B grants R and I rights to C.



User	Table	Grantor	Read	Insert . . .	Copy
D	*Y*	*A*	5	0	yes
B	*Y*	*A*	10	10	yes
C	*Y*	*B*	15	15	yes
B	*Y*	*D*	20	0	yes
C	*Y*	*B*	25	25	yes .

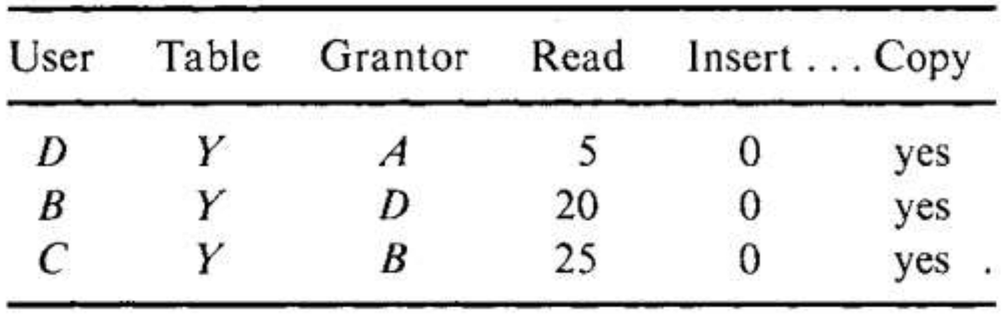
- What if A revokes B's rights at t=40

Grant/Revocation of Rights



User	Table	Grantor	Read	Insert . . .	Copy
D	*Y*	*A*	5	0	yes
B	*Y*	*A*	10	10	yes
C	*Y*	*B*	15	15	yes
B	*Y*	*D*	20	0	yes
C	*Y*	*B*	25	25	yes .

- What if A revokes B’s rights at t=40
  - Second and third rows should be deleted.
  - R rights of B shall be preserved as these were granted by D.
  - 5th row is modified as I right of C was through B and not via D (indirectly)



User	Table	Grantor	Read	Insert . . .	Copy
D	*Y*	*A*	5	0	yes
B	*Y*	*D*	20	0	yes
C	*Y*	*B*	25	0	yes .

Confused Deputy Problem

Confused Deputy

- Fortran Compiler Installed in a directory say SYSX
- Writes billing to a file called SYSX/BILL
- Writes statistics to a file called SYSX/STATS
- SYSX directory is privileged (only the compiler can write into it because it had a LISENCE file)
- Usage of the Fortran compiler will look like this:
- SYSX/FORT <input file> <output_file>
- What happens when user issues a command
  - SYSX/FORT <input file> SYSX/BILL
  - SYSX/BILL is overwritten (why?)

Confused Deputy

- Compiler is deputy and serving two masters
  - Invoker yields his authority to the compiler when he says "RUN SYSX/ FORT".
  - The other authority of the compiler stems from its homefiles license.
- Compiler carries some authority from each to perform its respective duties. Check against which authority?
  - Statistics/billing: authority granted by its home files
  - Output file: authority from its invoker.
  - By naming output file as SYSX/BILL, invoker has tricked compiler in granting rights of license file to invoker.
  - Compiler?
- Should the compiler check and prevent access to director/output file
  - name SYSX was not invented at the time of writing the code
  - Many sensitive files could be in use. Generic solution?

Confused Deputy

- One possible solution : Switching hats
- The compiler wears two hats
  - One hat when sensitive information like the file BILL was written
  - Other hat based on user's privileges to write user file
- How to generalize this ina large complex program,?
- A program may require multiple hats
- ACL not suited for mitigating confused deputy.

Confused Deputy

- One possible solution : Switching hats
- The compiler wears two hats
  - One hat when sensitive information like the file BILL was written
  - Other hat based on user’s privileges to write user file
- How to generalize this ina large complex program,?
- A program may require multiple hats
- ACL not suited for mitigating confused deputy.

Confused Deputy

- A computer program that is fooled into misusing authority leading to a privilege escalation
- =

Why Capabilites?

- Compiler is an online and paid (pay per use) service
- If User supplies SYS/BILL as output file, ACL based system allows system to open file on user's request.
- Capability based system shall check user's rights for SYS/BILL and shall not allow this operation.

	Compiler	Input File	Output File	SYS/BILL
User	Invoke	R, W	R, W	
Compiler		R	W	W
Service Provider				R

Examples

- CSRF (Cross Site Request Forgery)
  - Consider two websites normal.com and malicious.com
  - User is logged into normal.com and session is maintained by cookies stored on his machine. is logged in and his session is being maintained by cookies.
  - The attacker has placed HTML code (a request to normal.com for some activity) on malicious.com
  - User is somehow lured to malicious.com and click a button that invokes request to normal.com with user's credential stored as cookie..
  - No authorization check at normal.com.
- CSS (Cross Site Scripting)
- Click-jacking
- FTP Bounce

Information Flow Control

Information Flow

- Access control mechanisms
  - Check and regulate access of objects,
  - No control on what subjects might do with the information contained in the objects.
  - May lead to information "leakage"
- Flow controls are concerned with
  - the right of dissemination of information,
  - specify valid channels along which information may flow.

Information Flow

- Two security classes A and B
- Information is permitted to flow
  - within a class
  - Upward (class with higher security clearance)
- Information is not allowed to flow
  - Downward
  - or to unrelated classes
- $A \leq B$, class A information is lower than or equal to class B information.
- Class A information is permitted to flow into class B but not vice versa.

Examples: Information Flow

- Copy file1 to file2: flow of information from file1 to file2
- Assignment statement y=x;
  - information flow from x to y. No flow of value of X is known.
- Statement y = x/10: some information flow from x to y
- Statements z = x; y = z;
  - Indirect flow from x to y through z; direct from from x to z
  - No flow from z to y, y can't be used to deduce initial value of z
- Statement z = x + y
- Statement z = x xor y

Security and Precision

- F = set of all possible flows in an information flow system,
- P = set of all flows authorized by a given flow policy
- E = set of "executable” flows as per flow control mechanisms
- P ⊆ F and P ⊆ F
- System is
  - secure if E ⊂ P; all executable flows are authorized.
  - precise if E = P; all authorized flows are executable.

Information Flow Channels

- ❑ Legitimate Channels
  - intended for information transfer between processes e.g., the parameters of a procedure.
- ❑ Storage Channels
  - objects shared by more than one process or
- ❑ Covert Channels
  - which are not intended for information transfer at all example power consumption, memory access patterns, timing channel (program run time proportion to some secret)
- ❑ Legitimate channels are the simplest to secure.
- ❑ Securing storage channels - file, variable, and status bit etc. must be protected.

Information Flow Models

Bell LaPadula Model

- Focus: data confidentiality, controlled access to information
- To allow a specific access, the clearance of a subject is compared to the classification of the object
- Security rules : Two MAC and one DAC
  - Simple Security Property: a subject at a given security level may not read an object at a higher security level. (no read up)
  - Star (*) Property: a subject at a given security level may not write to any object at a lower security level. (no write down)
  - Discretionary Security Property uses an access matrix. Allows transfer of sensitive information to trusted subjects.
- Read Down, Write Up
- Limitations:
  - Data integrity; Covert channels; Network of systems
  - Suited only where security levels do not change dynamically

Bell LaPadula Model

- A lieutenant can ask a private to reveal all he knows and then copy this information into a general’s file without violating security.
- Consider an IT company: clerks have security level 1, programmers have security level 3, and company CEO has security level 5. A programmer can query a clerk about the company’s future plans and overwrite the CEO’s files that contain corporate strategy.
- Problem:
  - Data integrity not considered

BIBA (Integrity) Model

- Access control rules designed to ensure data integrity.
- Preservation of data integrity has three goals:
  - Prevent data modification by unauthorized parties
  - Prevent unauthorized data modification by authorized parties
  - Maintain internal and external consistency (i.e. data reflects the real world)
- Data and subjects are grouped into ordered levels of integrity.
  - subjects may not corrupt data in a higher level
  - Data corruption from a lower level than the subject not allowed.

Thank you.