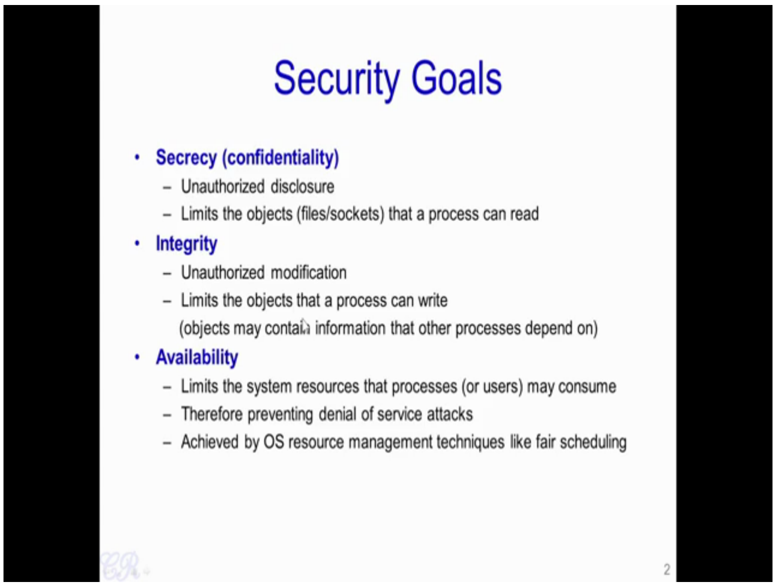
Introduction to Operating Systems
Prof. Chester Rebeiro
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week – 08
Lecture – 35
Operating System Security

Hello. In this video we will look at Operating Systems Security. Now, security as such is become extremely important in this current world because systems as such are always connected as a result it becomes very easy for malicious programs to enter into the system. Therefore, operating systems should be designed in such a way so as to prevent as much as possible any loss of information due to such malicious intuitions.

So, we will see a brief introduction to Operating Systems Security in today's video.

(Refer Slide Time: 01:00)



Security Goals

- **Secrecy (confidentiality)**
  - Unauthorized disclosure
  - Limits the objects (files/sockets) that a process can read
- **Integrity**
  - Unauthorized modification
  - Limits the objects that a process can write
    (objects may contain information that other processes depend on)
- **Availability**
  - Limits the system resources that processes (or users) may consume
  - Therefore preventing denial of service attacks
  - Achieved by OS resource management techniques like fair scheduling

2

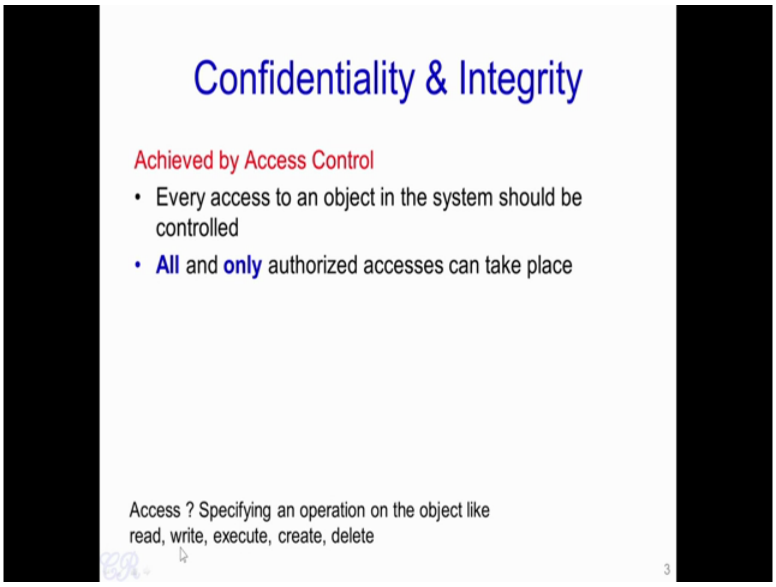
So, whenever we design a system, we need to have some security goals and these goals are the secrecy, integrity and availability of the system. Now, secrecy or confidentiality means that certain objects should not be visible to particular processes, for instance depending on the privilege of a process and also on the privilege on the user running that process certain objects like certain files stored in the desk or certain network sockets should not be readable by that process. Essentially what confidentiality achieves is that unauthorized disclosure of some objects should be prevented. What secrecy or confidentiality achieves is that unauthorized disclosure of certain objects like files and sockets should not happen.

With respect to integrity means unauthorized modifications. In other words a malicious user or a user with not sufficient privileges should not be able to write to a particular object, such as a file or a socket. To take an example, a normal user in the system should not be able to modify some system related files. Availability, on the other hand means that a particular user or a particular process has a limited use of the system resources. In other words, one particular user should not be able to hog the entire system resources.

If such a thing is not available in the system it could be lead to what is known as denial service attacks, wherein some malicious programs running on the system will prevent any other program which is present from having access to certain hardware resources like the CPU, the RAM or disk. So, availability in a system largely depends on how the operating system gets designed.

For instance, we have seen how CPU schedulers can be designed in order to ensure fair scheduling that is every process in the system gets a fair share of the CPU depending on it is priority. Now, in order to achieve secrecy and integrity what operating systems generally do is to have access control. So, we will be looking at access control techniques in this particular video.

(Refer Slide Time: 04:04)



Confidentiality & Integrity

Achieved by Access Control

- Every access to an object in the system should be controlled
- **All** and **only** authorized accesses can take place

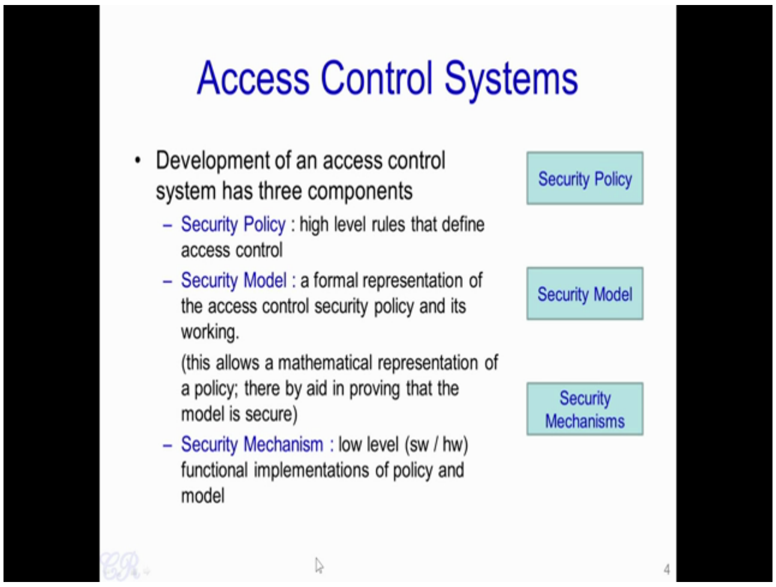
Access ? Specifying an operation on the object like read, write, execute, create, delete

3

In order achieve access control, what it means is that every access to an object such as a file, a socket or it could be a hardware device like a printer or a monitor should be controlled. Essentially, if a process wants to get access to a particular file or any other object in the system it has to go through an access control mechanism. Essentially, all and only authorized accesses can take place. In other words, if a process needs to access an object then it definitely should be given access to that particular object. So, what do we mean by access essentially by access we mean an operation on the object which a particular process intense to do.

So, these operations could be one of read, write, execute, create or delete. Now, this access should be permitted if and only if the particular process is authorized to make that access. In other words, only if a process is authorized to execute a particular program or read or write to a particular file only then should the operation be permitted.

(Refer Slide Time: 05:39)



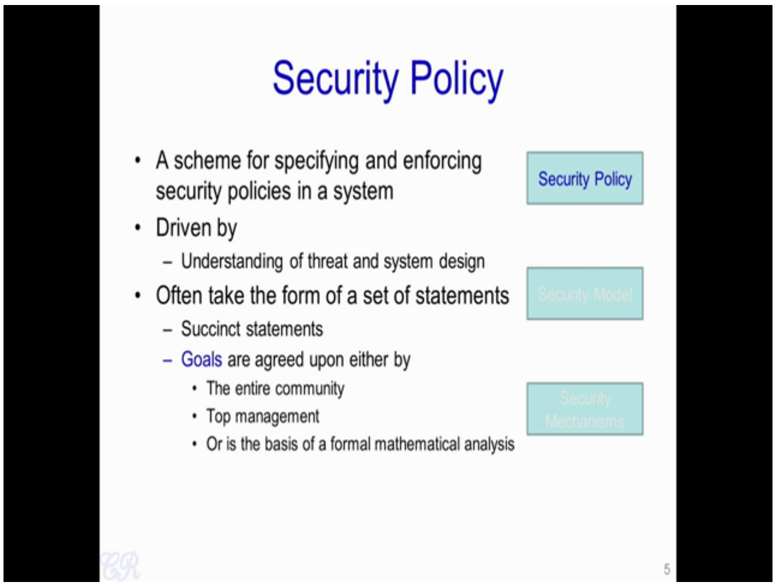
Access Control Systems

- Development of an access control system has three components
  - Security Policy : high level rules that define access control
  - Security Model : a formal representation of the access control security policy and its working.
    (this allows a mathematical representation of a policy; there by aid in proving that the model is secure)
  - Security Mechanism : low level (sw / hw) functional implementations of policy and model

In order to develop an access control systems there are three components security policies, security model and security mechanism. Security policies are high level rules that define the access control. Security model is the formal representation of the access control security policy and it is working essentially this is a formal model or a mathematically represented model of the security policy, and this model is used to help in proving various things about the system, for instance you could use this particular security model to prove that a particular system is secure.

Now, security mechanism is low level hardware or software that has the functional implementation of the policy that is the security policy and the model. So, in other words you can think of this as the high level rules that define the security policies or the security features that need to be supported by the system. This is a mathematical model or slightly lower level representation and while this mechanism is the actual implementation of the policies. So, let us look at each of these more in detail.

(Refer Slide Time: 07:10)



Security Policy

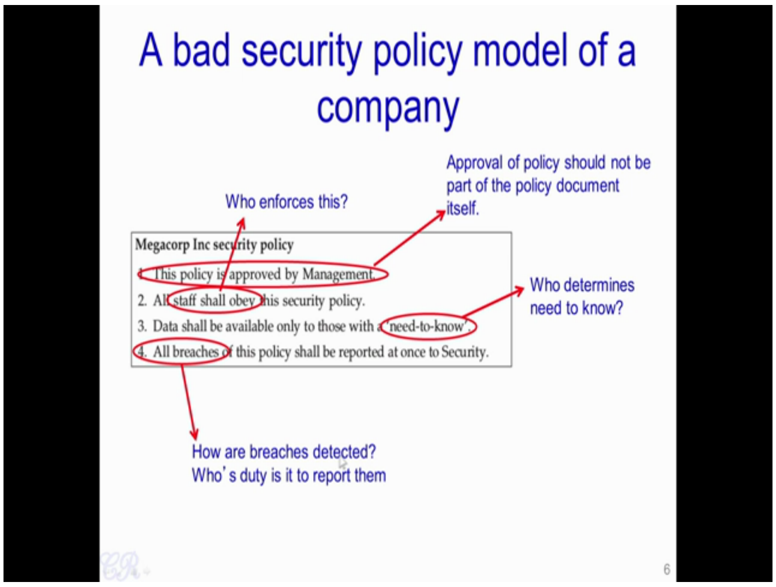
- A scheme for specifying and enforcing security policies in a system
- Driven by
  - Understanding of threat and system design
- Often take the form of a set of statements
  - Succinct statements
  - Goals are agreed upon either by
    - The entire community
    - Top management
    - Or is the basis of a formal mathematical analysis

Security Policy
Security Model
Security Mechanisms
5

So, security policy is essentially a scheme for enforcing with some policies within the system. How do we decide upon what policies a system should have, this is defined by what threats are present in the system as well as how the system is designed. So, developing a security policy is not easy. It would require a lot of brain storming to actually identify what are the various threats that or what are the various attacks that are possible on the system and once these attacks are known then policies are created in order to prevent such attacks. So, how is the security policy actually looks like?

A security policy essentially would be a set of statements. These statements should be extremely succinct and precise and should have the goals of the policy. So, these goals should be agreed upon by either the entire community or the entire set of people who are developing that system, or by certain top management and this forms the basis for the security model which is the formal mathematical representation of the policy.

(Refer Slide Time: 08:42)



A bad security policy model of a company

Megacorp Inc security policy

1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a "need-to-know".
4. All breaches of this policy shall be reported at once to Security.

Who enforces this?
Approval of policy should not be part of the policy document itself.
Who determines need to know?
How are breaches detected? Who's duty is it to report them
6

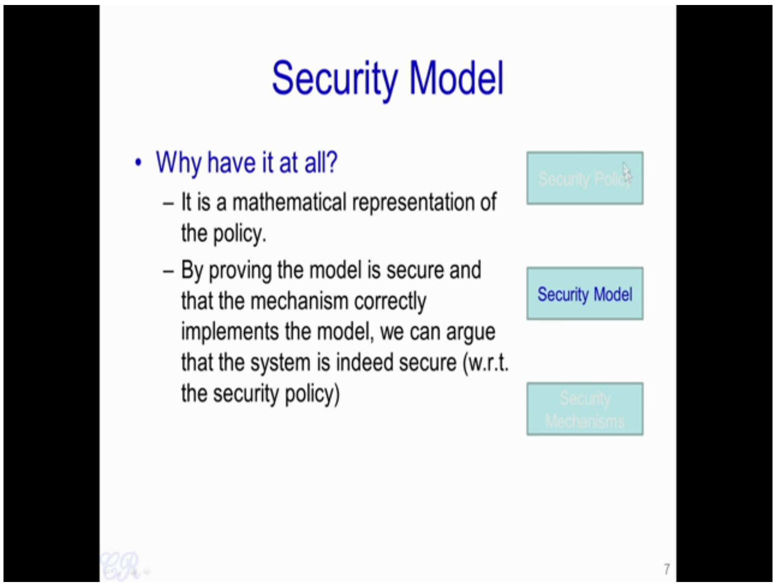
Let us take how a security policy should not be written. So, what we are seeing now is that security policies are not necessarily for computer systems, but it could also be for organizations. So, this particular security policy is for this particular company called Megacorp Incorporated. So, the security policy is extremely short, it has a just four statements and it reads as follows; this policy is approved by the management, all staff shall obey the security policy, data shall be available only to those with a need-to-know, all breaches of this policy shall be reported at once to security.

What is the issue with this particular security policy? If we actually take closer look at this policy from this particular company, what we see that there are lot of flaws in this. So, essentially the policy is not complete and cannot be used to build a security mechanism, for instance this policy is approved by management the first thing. So, typically in our security, policy the approval of the policy should not be part of the policy document itself.

Second, all staff shall obey. So, what we mean by this? Who enforces that a staff should obey? Is it moral requirement of the staff to actually obey to this policy or there is actually a unit which enforces all staff in the organization to obey for this particular policy? Third, the need-to-know, how or how to be defined what I need to know? Who determines who should know or what information? So, this is not specified or not very obvious from these particular policy statements.

Finally, all breaches of this policy should be reported at once the security. So, how are the breaches deducted? The security policy does tell not tell any anything of this form whose duty is to report them. This two is not mentioned over here. So, as you see even though this particular policy is very short and very tars and incorporates itself just four points, still it leads to a lot of ambiguity and such ambiguity would lead to a weak security for that company. So, when we have to write a security policy for a company it should be complete in all aspects.

(Refer Slide Time: 11:36)



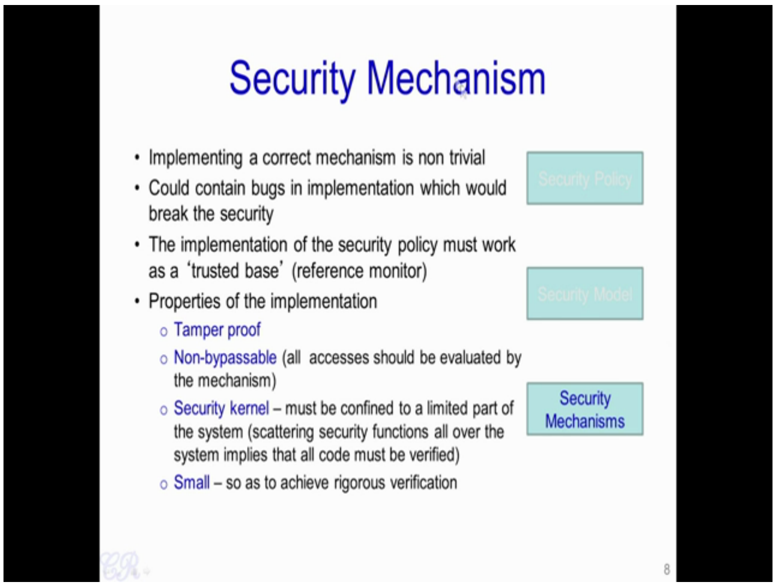
Security Model

- Why have it at all?
  - It is a mathematical representation of the policy.
  - By proving the model is secure and that the mechanism correctly implements the model, we can argue that the system is indeed secure (w.r.t. the security policy)

Security Policy
Security Model
Security Mechanisms
7

Now, let us look at the security model, essentially why do we need to have the security model at all? Why cannot we go from the security policy, use the security policy and directly implement security mechanisms. Why do we need to have a security model present in this entire hierarchy? Essentially by having a security model we are able to model the security policy in a lower and more formal construct. In this way any gaps in the security policy will be deducted and secondly, we could also use tools and techniques in order to argue or prove that the system is indeed secure with respect to the security policy that was defined.

(Refer Slide Time: 12:31)



Security Mechanism

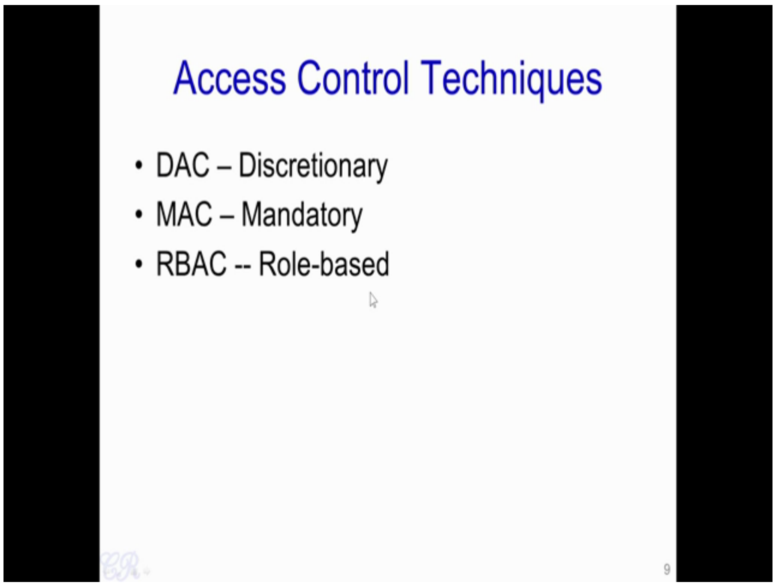
- Implementing a correct mechanism is non trivial
- Could contain bugs in implementation which would break the security
- The implementation of the security policy must work as a 'trusted base' (reference monitor)
- Properties of the implementation
  - Tamper proof
  - Non-bypassable (all accesses should be evaluated by the mechanism)
  - Security kernel – must be confined to a limited part of the system (scattering security functions all over the system implies that all code must be verified)
  - Small – so as to achieve rigorous verification

So, the last aspect is the Security Mechanism. Security Mechanism as we have seen deals with implementing of the security policy. Now, it is important that implementing the security mechanism is bug free, it has no bugs. Why is it extremely important that there are no bugs? This is because if there are bugs in the implementation of the mechanism then it would be possible for attackers to exploit that bugs and get in unauthorized accesses into that system.

Second, the implementation of the security mechanism should be the trusted base. Essentially this is the core from where the security of the entire system would depend on therefore, if it itself is buggy and incomplete it will not create a secure system. So, properties of the security mechanism implementation is that it should be tamper proof, non-by passable and by non-by passable, we mean that all accesses into the system should first be evaluated by the security mechanism then it should be a security kernel. So, what it means by this is that the entire implementation for the security policy should be confined to a limited part of the system and it should be scattered into various parts of the system.

So, having a security kernel where everything is just confined into, say a small part of the entire system would make it easy to test and debug and verify in terms of security. Again it should also be small. So, a small size for the security mechanism will ensure that it can undergo some rigorous verification.

(Refer Slide Time: 14:35)

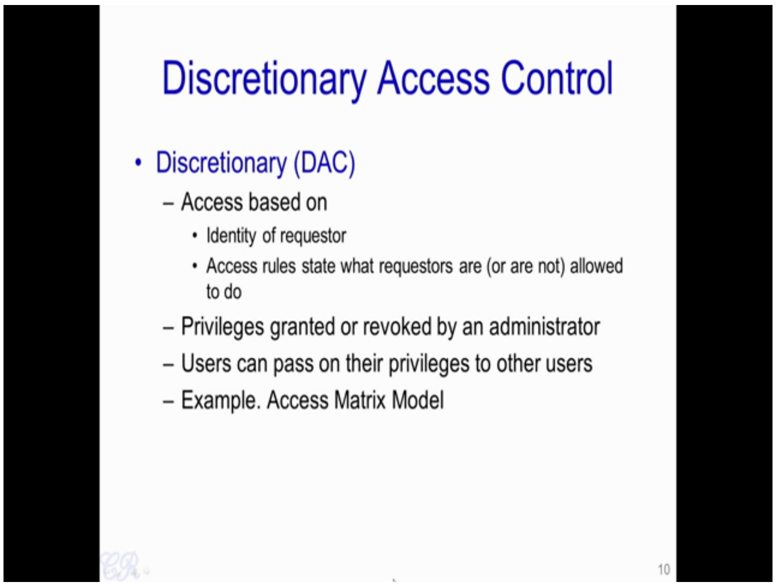

Access Control Techniques

- DAC – Discretionary
- MAC – Mandatory
- RBAC -- Role-based

9

Now, there are 3 techniques for access control. They are the DAC, MAC and the RBAC. So, DAC is the Discretionary Access Control, MAC is Mandatory Access Control and RBAC is Role-based Access Control. So, we will be looking at the DAC and MAC at this particular lecture.

(Refer Slide Time: 15:00)


Discretionary Access Control

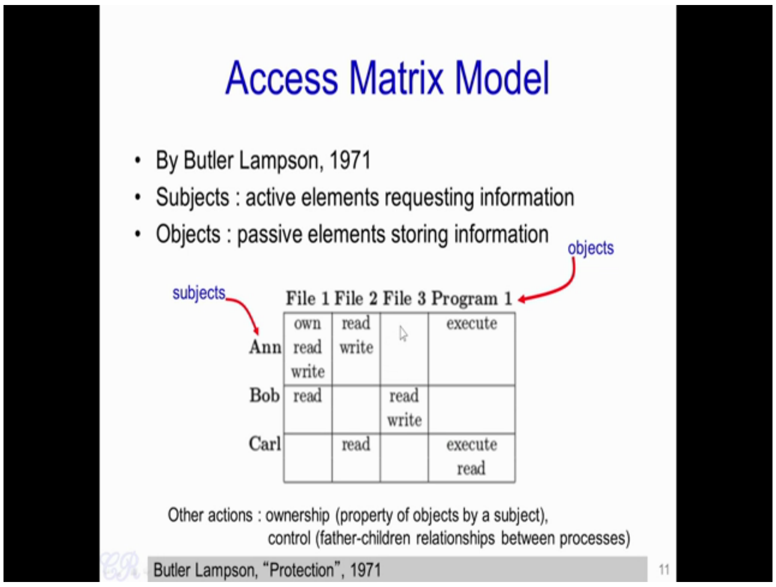
- Discretionary (DAC)
  - Access based on
    - Identity of requestor
    - Access rules state what requestors are (or are not) allowed to do
  - Privileges granted or revoked by an administrator
  - Users can pass on their privileges to other users
  - Example. Access Matrix Model

10

Discretionary Access Control or DAC is an access control which is based on the identity of the requester. There would be a set of access rule what requesters are or not allowed to do. So, these access rules would define essentially what objects the requester, it also known as the subject could access and what objects could be read, written to, executed, created or deleted and so on.

Now, the privileges for these various objects are granted as well as revoked by the system administrator. So, users if they have a particular privilege then they can also pass on that privilege to other users. A very common example of a DAC system that is the discretionary access control system is the access matrix model.

(Refer Slide Time: 16:05)



Access Matrix Model

- By Butler Lampson, 1971
- Subjects : active elements requesting information
- Objects : passive elements storing information

objects
subjects

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

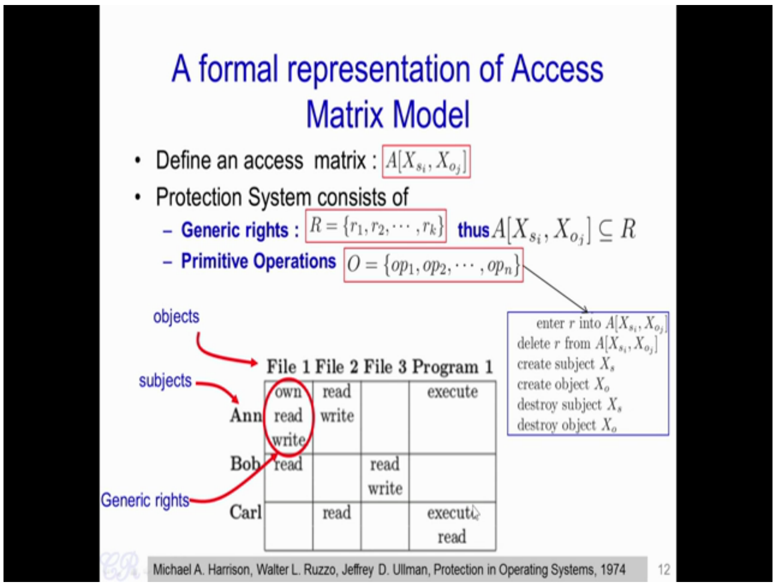
Other actions : ownership (property of objects by a subject), control (father-children relationships between processes)

Butler Lampson, "Protection", 1971 11

So, the Access Matrix Model was designed Butler Lampson in 1971. It comprises of subjects and objects, in addition to this it also has a table in this form where each cell has some particular actions. So, subjects are active elements in the system, such as users of the systems or processes or programs that are requesting information.

Objects on the other hand are passive elements which, for instance stores information now the actions are specified in each cell of this particular matrix. So, for instance the subject; Ann has execute permissions for program 1, she has read and write programs permissions for file 1 and she has read and write permission for file 1 as well as she is the owner for file 1. So, if you look more carefully, Ann does not have any permission on file 3, she can neither read it nor write it nor she can actually read or write the program 1.

(Refer Slide Time: 17:27)



A formal representation of Access Matrix Model

- Define an access matrix : $\boxed{A[X_{s_i}, X_{o_j}]}$
- Protection System consists of
  - **Generic rights :** $\boxed{R = \{r_1, r_2, \cdots, r_k\}}$ **thus** $A[X_{s_i}, X_{o_j}] \subseteq R$
  - **Primitive Operations** $\boxed{O = \{op_1, op_2, \cdots, op_n\}}$

enter r into $A[X_{s_i}, X_{o_j}]$
delete r from $A[X_{s_i}, X_{o_j}]$
create subject X_s
create object X_o
destroy subject X_s
destroy object X_o

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Michael A. Harrison, Walter L. Ruzzo, Jeffrey D. Ullman, Protection in Operating Systems, 1974

12

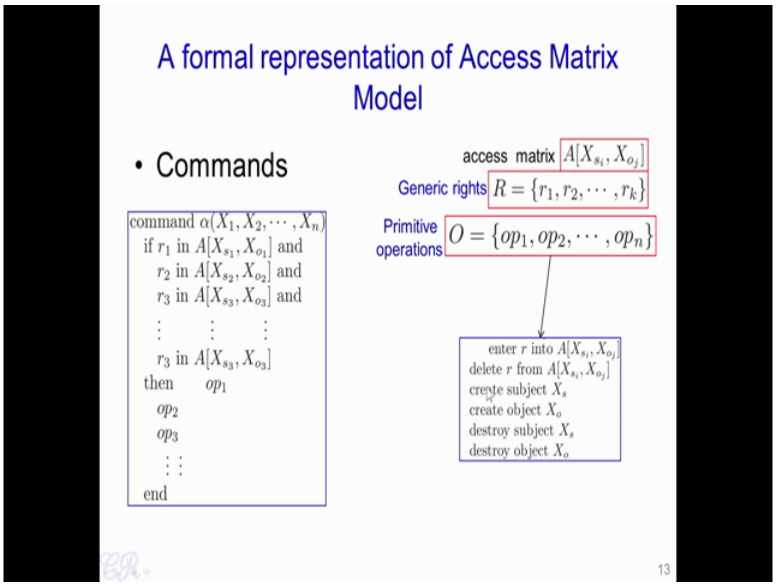
So, in order to represent this access matrix model in a formal method, what we can do is define this matrix A which comprises of X s i that is for subjects and X o j, which is for an object. So, A of this particular thing would indicate the subject s i, for instance Bob and an objects for instance file 2. Then we could also give rights to each cell in this particular matrix.

For example the rights could be taken from a set like this r 1, r 2 to r k and this particular thing A of X s i, X o j is the subset of R. So, for example, the rights in this example is own read write execute, so on and these would be in the set R and corresponding to each cell in this matrix we define what the rights are. So, this would define what the right for subject s i has on the object o j. In addition to this there is something known as the primitive operations. So, this is represented by O and it is the set of 6 operations which is specified here.

So, this is for instance enter r into some location in the matrix or delete r from that location here r is taken from the generic rights. So, it is an element form the set r similarly other operations are to create a particular subject X s create an object X o destroy a subject X s or destroy an object X o.

These are the only primitive operations that can be done on these this particular access matrix. So, as we see each of these primitive operations would either modify the contents of a particular cell in the matrix or it could delete an entire column with the destroy subject can destroy object. It could either delete a particular column or a particular row or if it wants it could also create a new column that is creating an object or creating a subject. Once, we have defined such primitive operations we can define something which is a bit more complex and paste on this primitive operations which are known as commands.

(Refer Slide Time: 20:17)



A formal representation of Access Matrix Model

- Commands

```
command $\alpha(X_1, X_2, \cdots, X_n)$
  if $r_1$ in $A[X_{s_1}, X_{o_1}]$ and
    $r_2$ in $A[X_{s_2}, X_{o_2}]$ and
    $r_3$ in $A[X_{s_3}, X_{o_3}]$ and
    $\vdots$    $\vdots$    $\vdots$
    $r_3$ in $A[X_{s_3}, X_{o_3}]$
  then    $op_1$
    $op_2$
    $op_3$
     $\vdots$ $\vdots$
  end
```

access matrix $A[X_{s_i}, X_{o_j}]$

Generic rights $R = \{r_1, r_2, \cdots, r_k\}$

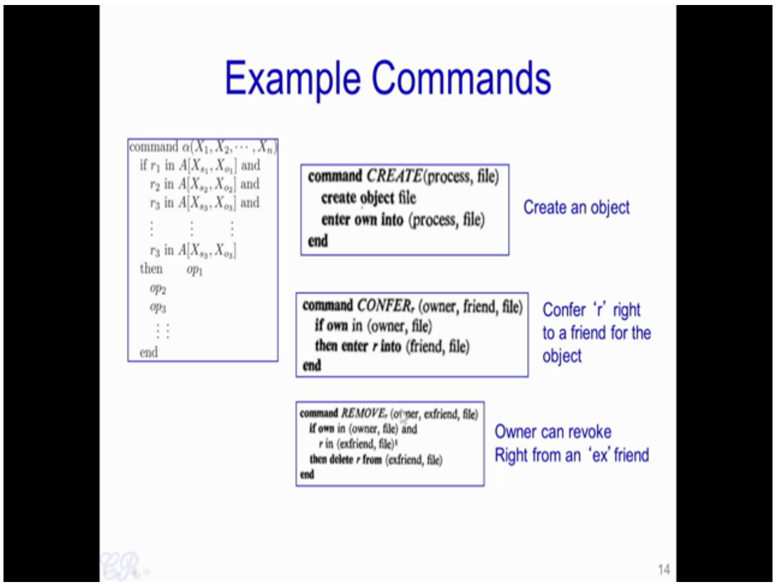
Primitive operations $O = \{op_1, op_2, \cdots, op_n\}$

enter r into $A[X_{s_i}, X_{o_j}]$
delete r from $A[X_{s_i}, X_{o_j}]$
create subject X_s
create object X_o
destroy subject X_s
destroy object X_o

13

So, to recollect we have the access matrix which is specified by A. We have the generic rights r 1 to r k and we have six primitive operations as we specify here. So, we can use all of this to create complex commands like this way a typical command would look like this and it would be called say, for instance alpha which takes several parameters x 1 to x n and we can have a set of rules, for instance if r 1 is in this matrix cell and r 2, r 3 and so on then perform these following operations. So, what we are saying is that based on these low level primitive operations, the genetic rights and the access matrix, we can have more complicated commands on the access matrix.

(Refer Slide Time: 21:19)



Example Commands

```
command $\alpha(X_1, X_2, \cdots, X_n)$
  if $r_1$ in $A[X_{s_1}, X_{o_1}]$ and
    $r_2$ in $A[X_{s_2}, X_{o_2}]$ and
    $r_3$ in $A[X_{s_3}, X_{o_3}]$ and
    $\vdots$      $\vdots$      $\vdots$
    $r_3$ in $A[X_{s_3}, X_{o_3}]$
  then    $op_1$
    $op_2$
    $op_3$
     $\vdots$ $\vdots$
  end
```

```
command CREATE(process, file)
  create object file
  enter own into (process, file)
end
```

Create an object

```
command CONFER_r (owner, friend, file)
  if own in (owner, file)
  then enter r into (friend, file)
end
```

Confer 'r' right to a friend for the object

```
command REMOVE_r (owner, exfriend, file)
  if own in (owner, file) and
    r in (exfriend, file)
  then delete r from (exfriend, file)
end
```

Owner can revoke Right from an 'ex' friend

14

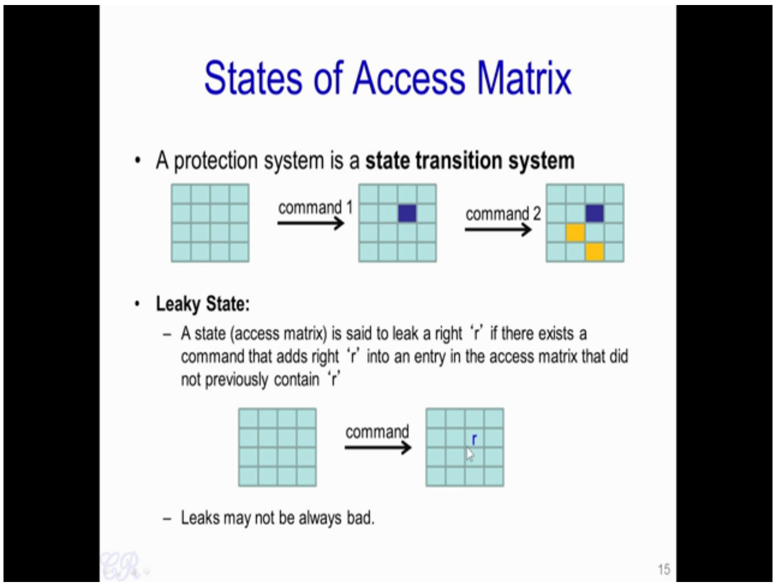
Let us take a few examples of such commands, for instance create a process comma file. So, process here is the subject and what it means to say is this process wants to create this file. So, the command will work like this. So, first we use the primitive operations create an object file and then enter own into that process comma file. So, this process comma file corresponds to the cell in the access matrix. If this particular command is not present then it will not be possible for a process to actually create a particular file. So, this is a very simple example.

We could have something a bit complicated like this particular example that is CONFER r - owner, friend, file. So, owner and friend are the subjects and file is the object over here and what is intended to do by this particular command on the access matrix is that, we want to confer write r which is present over here to a friend for that particular object. So, this owner wants to confer the right r for the friend with respect to this particular file. So, what goes on in this particular command, essentially we first test if the owner is the indeed the owner of that file. So, we see that first if own right is present in owner comma file in the matrix cell corresponding to owner the subject and file the object if this is indeed true then enter r that is enter the right r into the front comma file part of the matrix.

Similarly, we can have another example where we can remove a right from an exfriend we are passing this particular command three parameters the owner ex friend and the file.

So, essentially what we want to do is revoke the particular right r from this exfriend which is another subject. So, the command looks like this if own is in the owner comma file that is if owner is indeed the owner of the file and the right r is present in exfriend comma file then deletes r from exfriend comma file. Therefore, we can remove the right r which could be a read or write or execute right on the file with respect to exfriend essentially the exfriend subject will not have the right r on the particular file.

(Refer Slide Time: 24:24)



States of Access Matrix

- A protection system is a **state transition system**

command 1
command 2

- **Leaky State:**
  - A state (access matrix) is said to leak a right 'r' if there exists a command that adds right 'r' into an entry in the access matrix that did not previously contain 'r'

command
r

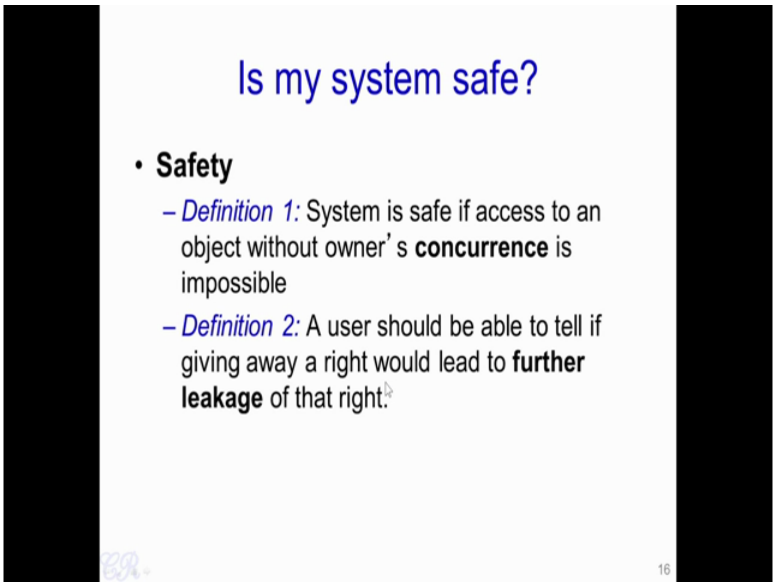
  - Leaks may not be always bad.

15

Protection system, if we actually represent it by this particular matrix and what we seen is that we could apply commands to it like we have seen a couple of commands in the examples in the previous slide and with each command we are modifying this particular access matrix. So, this command, for instance has modified this cell over here in the access matrix. Similarly, if you run another command, another part of the access matrix or multiple cells in the access matrix will be modified. So, based on this we define, what is known as a leaky state? A state of the access matrix is said to leak right r if there exist a command that adds the right r into an entry in the access matrix that did not previously contain r.

So, what this means is that we have the state of the matrix here and we run a command and as a result there is an r or right r which is entered into this particular matrix. So, what we say is that this particular state of the matrix leaks r. Now, a leaky state does not always need to be bad. So, it is meant to left to us to actually determine whether leaking this r from a particular cell in the matrix is good or bad with respect to security of the system.

(Refer Slide Time: 26:03)



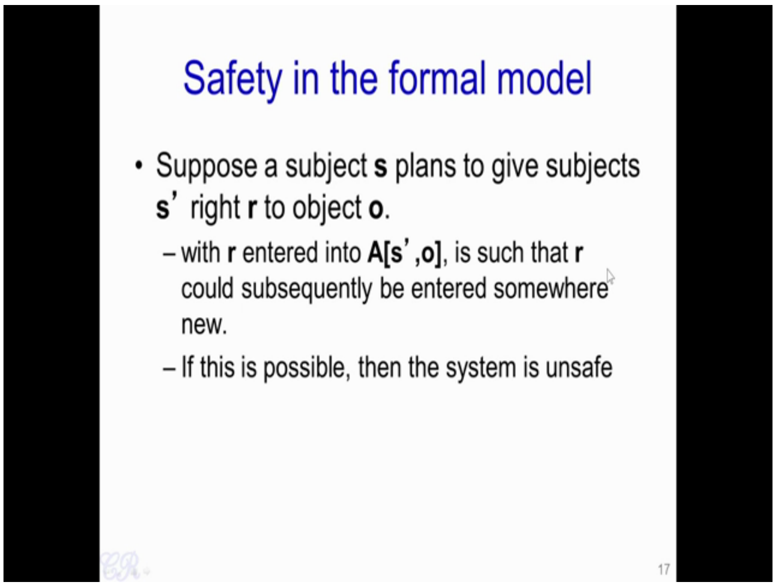
Is my system safe?

- **Safety**
  - *Definition 1:* System is safe if access to an object without owner' s **concurrence** is impossible
  - *Definition 2:* A user should be able to tell if giving away a right would lead to **further leakage** of that right.

16

Let us define when a system is said to be safe. So, we have two definitions of safety. The first definition; a system is safe if access to an own in an object without the owner's concurrence is impossible. Second, a user should be able to tell if giving away a right to a particular subject and obstruct with respect to an object would lead to further leakage of that right.

(Refer Slide Time: 26:43)



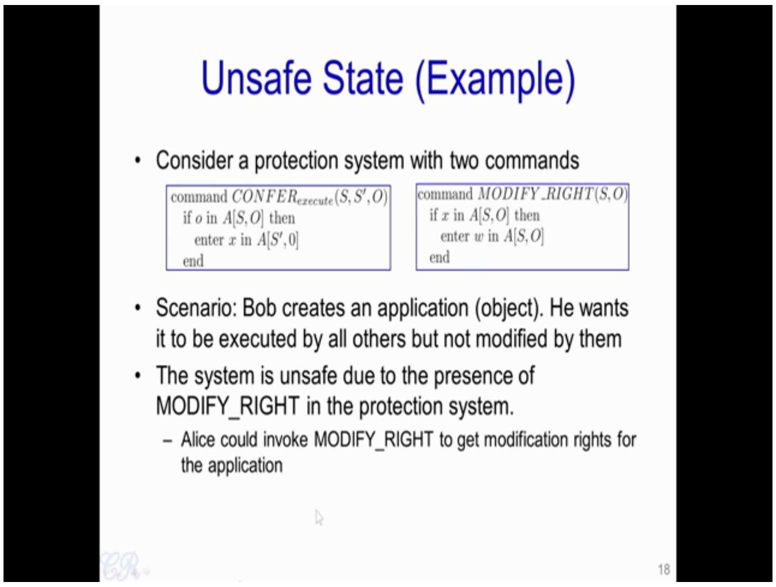
Safety in the formal model

- Suppose a subject **s** plans to give subjects **s'** right **r** to object **o**.
  - with **r** entered into **A[s' ,o]**, is such that **r** could subsequently be entered somewhere new.
  - If this is possible, then the system is unsafe

17

Let us see what this means form a formal model. Suppose, a subject s plans to give another subject s prime the right r to object o, with r entered in s prime comma o that is with new right added to this particular cell of the matrix. Is it possible that r could subsequently be entered somewhere else in the matrix A, if such a thing is possible then the system is set to be unsafe. So, essentially a system is unsafe if any operation or any command run on the access matrix could result in that particular right being conferred or transferred to someone else or to somewhere else in the matrix.

(Refer Slide Time: 27:47)



Unsafe State (Example)

- Consider a protection system with two commands

```
command $CONFER_{execute}(S, S', O)$
  if $o$ in $A[S, O]$ then
    enter $x$ in $A[S', 0]$
  end
```

```
command $MODIFY\_RIGHT(S,O)$
  if $x$ in $A[S,O]$ then
    enter $w$ in $A[S,O]$
  end
```

- Scenario: Bob creates an application (object). He wants it to be executed by all others but not modified by them
- The system is unsafe due to the presence of MODIFY_RIGHT in the protection system.
  - Alice could invoke MODIFY_RIGHT to get modification rights for the application

18

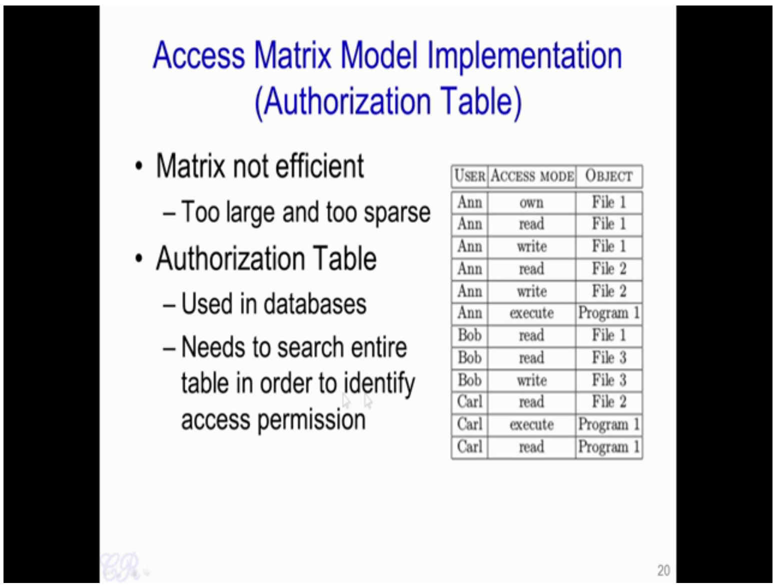
We will look at an example of an unsafe state. Let us consider these two commands one is the, CONFER execute. So, the subject S once we confer the subject S prime the, execute right for O. So, this command confer execute S comma S prime O states that a subject if it is the owner of the object O then it can give the right x that it can give the right to execute object O to another subject S prime.

Now, let us say our system also has this particular command which is called modified rights which states as follows, if a particular subject has an execute right with respect to an object then it can also enter a right in that particular object. In other words, what this modified right allows us to do is that if a particular subject can execute an object then it can actually modify its rights in order to change the object as well. So, essentially it can write into that object.

Let us look at a particular scenario to see, how this example shows an unsafe state. So, let us say, Bob creates an application object, he wants to be executed by all others, but not modified by them. So, the system is obviously, unsafe due to the presence of modified right in the protection system. Access Alice another person who has the execute right for that particular object or application could invoke modify right to get the modification rights on that application and once Alice gets the right to modify that application then there is nothing stopping Alice from actually changing that operation.

So, what we see is that because Bob has given the execute permission on a particular object, and the systems policies or the systems access control mechanism is built in such a way that because this right is present in a particular object then the w write is also given to that particular object essentially if a subject has the execute right for an object then that right is actually transformed and transferred and it will cause the w write to be also associated with that particular object with respect to that subject. Therefore, this particular state in the system is an unsafe state.

(Refer Slide Time: 30:48)



Access Matrix Model Implementation (Authorization Table)

- Matrix not efficient
  - Too large and too sparse
- Authorization Table
  - Used in databases
  - Needs to search entire table in order to identify access permission

USER	ACCESS MODE	OBJECT
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 3
Bob	write	File 3
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

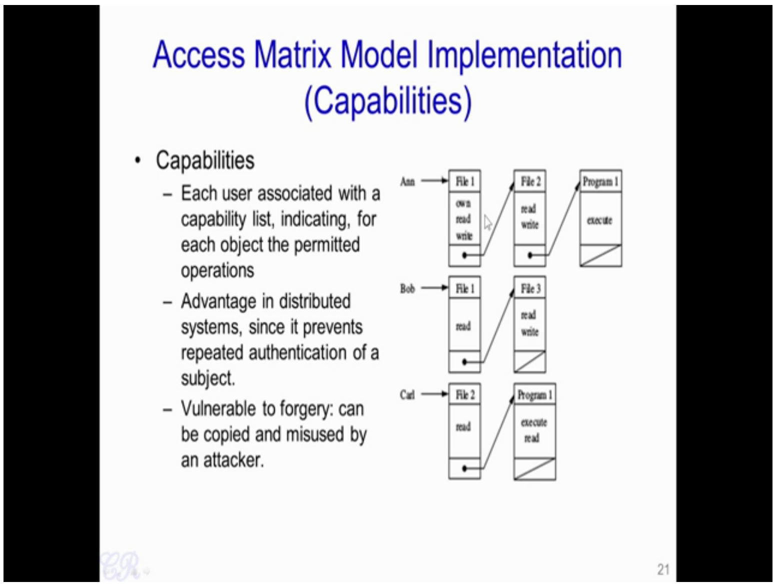
20

How do we actually implement the access matrix model? So, one way as we seen is to have the matrix representation; however, the issue with this particular representation is that it is too large as well as it is too sparse. So, it is not a very efficient way to use.

The other technique is used to something known as an authorization table as is shown over here. This particular table is used in data bases and what it actually helps is that each row comprises of a user it has a right and it has an object. So, it says that for instance and has a read write on the object file 1.

So, for every right that Ann holds there will be a separate row in this particular table. So, the problem with this particular scenario is that or this particular implementation is that one needs to at least search the entire table in order to identify whether a particular user has a particular right on a particular object. So, it again is having a lot of performance issues with respect to how long it takes to determine a particular write or a subject and object.

(Refer Slide Time: 32:14)



Access Matrix Model Implementation (Capabilities)

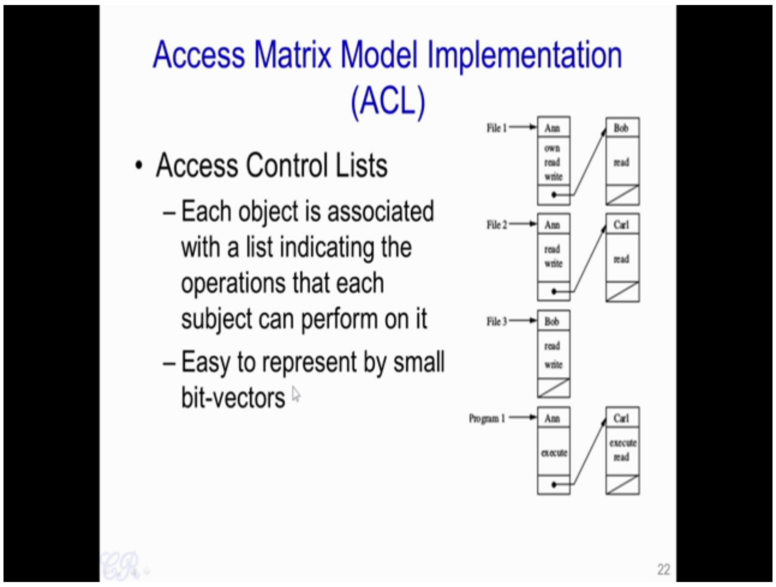
- Capabilities
  - Each user associated with a capability list, indicating, for each object the permitted operations
  - Advantage in distributed systems, since it prevents repeated authentication of a subject.
  - Vulnerable to forgery: can be copied and misused by an attacker.

Ann
File 1
own
read
write
File 2
read
write
Program 1
execute
Bob
File 1
read
File 3
read
write
Carl
File 2
read
Program 1
execute
read
21

Another technique of implementing the access matrix model is by using capabilities. Here, each user is associated with the capability list indicating for each object the permitted operations. For instance, Ann, for file 1 has the following rights over here, for file 2 she has these rights, program 1 she has these rights. Similarly, for every user we have such a list and these are the capabilities list for that user with respect to each of these objects.

The advantage of such a model or such an implementation is from a distributed system scenario since it prevents repeated authentication of a subject. So, essentially once, the subject authenticates itself in the distributed environment then it could obtain this entire list and the system will know it is rights for each object which is present. However, that limitation is that it is vulnerable to forgery if a particular right or a list is copied by an attacker it can be then misused.

(Refer Slide Time: 33:34)



Access Matrix Model Implementation (ACL)

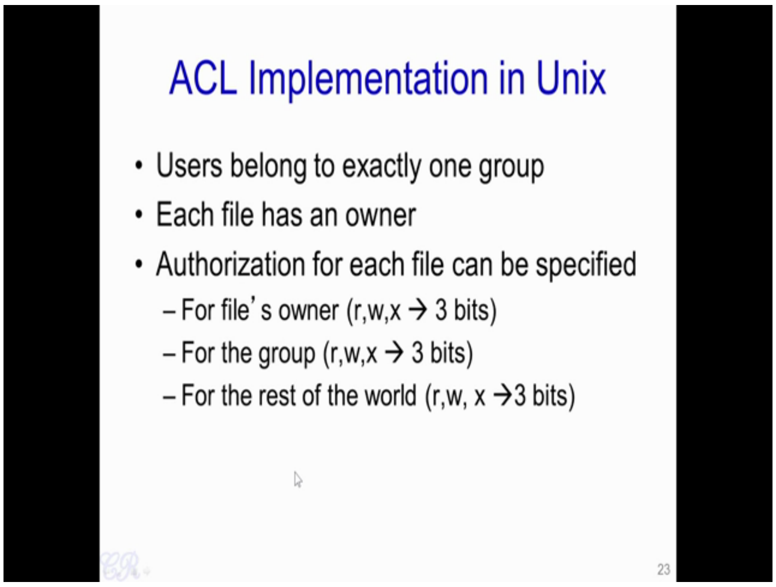
- Access Control Lists
  - Each object is associated with a list indicating the operations that each subject can perform on it
  - Easy to represent by small bit-vectors

File 1
Ann
own
read
write
Bob
read
File 2
Ann
read
write
Carl
read
File 3
Bob
read
write
Program 1
Ann
execute
Carl
execute
read

22

The third way of implementing the access control model is by what is known as ACL or Access Control List. Essentially, each object here is associated with the list indicating the operations that each subject can perform on it. Essentially, this is the opposite of the previous way. So, corresponding to object we have a list and each node in the list has a subject and the corresponding operations that are present here. So, the advantage that we get with this particular thing is that it is easy to represent by small bit factors. So, if you look at how UNIX actually implements this ACL.

(Refer Slide Time: 34:19)



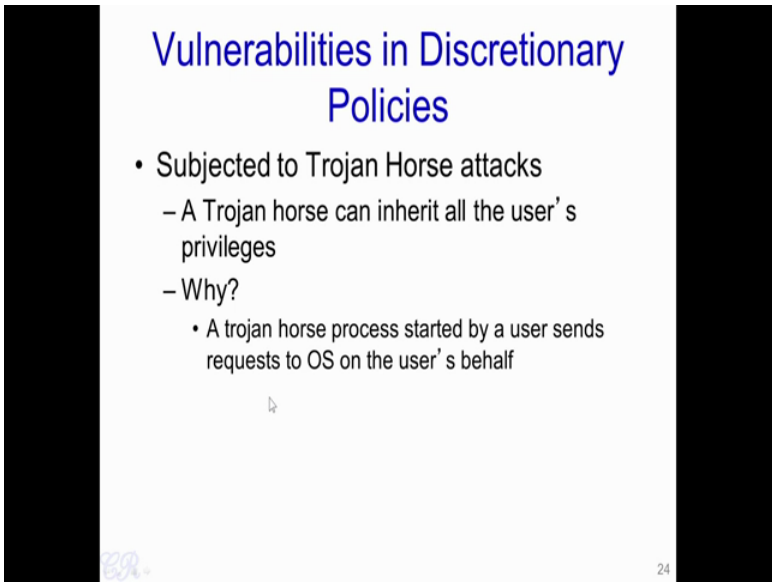
ACL Implementation in Unix

- Users belong to exactly one group
- Each file has an owner
- Authorization for each file can be specified
  - For file's owner (r,w,x → 3 bits)
  - For the group (r,w,x → 3 bits)
  - For the rest of the world (r,w, x →3 bits)

23

So, we see that every file in a unique system has 9 bits associated with it. So, this gives the authorization for each file to be specified, for instance there are 3 bits which specify read, write, execute for the file for the owner of the file. Then we have the group for which we have the 3 bits r, w, x for the group and for the rest of the world, for the others that is we have 3 more bits thus in a unique system which uses the ACL implementation of the access matrix. There are 9 bits involved and each of these bits would give permissions to various users of that file.

(Refer Slide Time: 35:13)



Vulnerabilities in Discretionary Policies

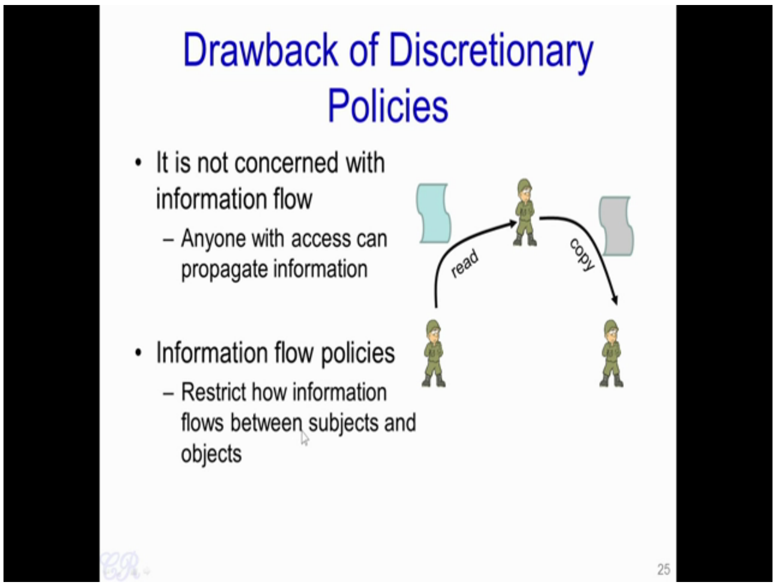
- Subjected to Trojan Horse attacks
  - A Trojan horse can inherit all the user's privileges
  - Why?
    - A trojan horse process started by a user sends requests to OS on the user's behalf

24

Now, the vulnerabilities in the discretionary policy are that, it is subjected to the Trojan Horse attacks. What we mean by Trojan horse is that it is a small code which is present in a larger non-malicious code. So, this small Trojan present in the larger code is the malicious code; the issue with this is that because the Trojan is actually like a virus which is joined with a much larger program.

The Trojan can actually inherit all the users or all the processes privileges, for instance if you have a Trojan which is connected to let say a program which has a super user privileges to execute or which needs to execute as a route then the Trojan itself would be able to inherit or all the route permissions. The reason why this happens is that the Trojan present in the process would send request to the operating systems in the valid user behalf. From the OS prospective, the operating systems will not be able to distinguish whether it is from the valid process, which is started by a valid user or whether the request is coming from a Trojan. Therefore, it becomes difficult to detect.

(Refer Slide Time: 36:51)



Drawback of Discretionary Policies

- It is not concerned with information flow
  - Anyone with access can propagate information
- Information flow policies
  - Restrict how information flows between subjects and objects

read
copy
25

Another drawback of discretionary policy is that it is not concerned with information flow. Anyone with access to information can also propagate information, for instance if we have this one person over here who is able read a particular file there is nothing stopping this person from actually making copy of this file and transferring it to hundred different people. As such the discretionary policies do not take care of the flow of information from one person or one user to another. It is only capable of preventing access to information, but not the flow of information.

On the other hand, in order to consider flow of information also we need to use what is known as information flow policies which restricts how information flows between subjects and objects.

So, we will look at Information Flow Policies in the next video.

Thank you.