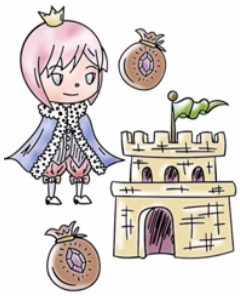
Access Control
Lesson Introduction

- Understand the **importance of access control**
- **Explore ways** in which access control can be implemented
- Understand **how access control** is implemented

So authentication helps us answer the question, who the request is coming from. But then we have to check if the source of the request is actually authorized to access the resource they're asking for. Well, this is the access control, or authorization problem. The focus of this lesson is going to be on access control and how it can be implemented in a computer system.

Controlling Accesses to Resources



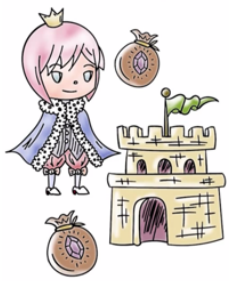
- **TCB** (reference monitor) sees a request for a resource, how does it decide whether it should be granted?
- **Example:** Should John's process making a request to read a certain file be allowed to do so?

We're going to talk about how we control access to resources. These are protected resources. Any program can make a request for such a resource, and we need to decide if the request should be allowed to go ahead, it should be granted, or we have to deny it.

So we've been talking about the importance of the trusted computing base. In fact, when someone is asking for a resource, they are generating a reference for that. They have to reference the resource they are interested in or they want to use, so the trusted computing base has to act as a reference monitor. No request for resource should go without the trusted computing base being involved in checking that request. Obviously, it'll have to see the source of the request is, what the target, or what resource is being requested. And then we're going to discuss how do we decide if it should be granted.

As a quick example, let's say John is a student in a class and there's a file that has the grades of all the students who are enrolled in that class. John, obviously, can make a request to read that file. Not only he wants to see his own grade, maybe he's curious about how other people did on a certain test or an exam. So John could make a request to read a file. In this case, we not only have to monitor the request that is going to come from John, but our access control that we going to do should decide that John should not be able to see other people's grades. So, in this case, the result may be denying the request. But the idea here is that references have to be monitored, and then we have to make some sort of a decision when a request comes, whether it should go ahead or it should be denied.

Controlling Accesses to Resources



- **Authentication** establishes the source of a request (e.g., John's UID)
- **Authorization** or access control answers the question if a certain source of a request (User ID) is allowed to read the file
- **Subject who owns a resource (creates it) should be able to control access to it (sometimes this is not true)**

So we talked about authentication. Authentication basically tells us when an application or a process makes a request, on whose behalf that request is being made.

So the example that we're just talking about, it's a process that John launched, then it is going to have John's User ID or UID. And authentication is how we know that this particular UID should be associated with the process. Authentication tells us about the source.

Authorization, or access control, which is the topic for this lesson, is going to answer the question knowing the source of a request, authentication is going to tell us that. Once we know the source of a request, in this case a User ID, and the target of the request, which is a file, and what they want to do with this resource is read this file, should we let them do it or not? So access control is similar to what we have in real world. You want to access something, and somebody has to decide, in this case, the TCB, or the reference monitor that is implemented by the TCB, whether that access is one that should be allowed to proceed or go forward, or it should not.

So let's quickly talk about how do we actually decide this. Resources that we have in the system, for example, files are created by certain users or subjects. You can, perhaps, think if Alice is creating a file, maybe she has the ability to decide who should be able to access it or not. The intention here being, it's her file, she chooses to decide who to share it with. So in many systems, actually, the idea of an owner of a resource is defined. This is the subject who creates the resource, and it is at the discretion of the owner how that resource can be shared. There are other kind of systems where this may not be the case and we're going to talk about that. For example, if you work for a company, the company may not allow you to decide how you can share certain sensitive data about that company.

Controlling Accesses to Resources

- **Access Control**
- Basically, it is about who is allowed to access what.
- Two parts
  - **Part I:** Decide who should have access to certain resources (access control policy)
  - **Part II:** Enforcement – only accesses defined by the access control policy are granted.
- **Complete mediation** is essential for successful enforcement

How do we control access to resources with a context of authentication that establishes a source and the idea that these authorization check or access control check has to happen if the resources have to be protected?

Really, for us to be able to answer this question we had, whether a request should be granted or denied, we have to know who is allowed to access what resources in a given system. So if you think about this, there are really two parts to this problem, or the way we're going to address this problem.

- **Part I:** Decide who should have access to certain resources (access control policy)

In part one, someone has to specify who has access to what. So this is called an access control policy. So there may be many ways to decide who gets to access what. There has to be a policy we have in place that is going to define this for us. So to answer this question about who can access what, is you tell the system how the resources should be shared, or who should be given access to what resource.

- **Part II:** Enforcement – only accesses defined by the access control policy are granted.

Once you do that, then it's the part two is about enforcement. The system has to monitor each request for these resources, and based on the policy that tells us who can access what, it should make sure any accesses that are allowed are consistent with the policy that we have. So enforcement basically says there is no way for you to go and access a resource when that access is not allowed by the policy. So define the policy that's in part one, and then enforce the policy, that's part two.

- **Complete mediation** is essential for successful enforcement

Now you can see the importance of complete mediation. Clearly complete mediation says, no one should be able to bypass and get to the resource without going through the trusted computing base. If you don't have complete mediation then we can't perform this check, access control check that we are talking about. That is possible only when the trusted computing base is involved in a request and that is why complete mediation is so important.

Access Control Matrix (ACM)

- An access control matrix (ACM) **abstracts the state relevant to access control.**
- Rows of ACM correspond to users/subjects/groups
- Columns correspond to resources that need to be protected.
- **ACM defines who can access what**
  - ACM [U,O] define what access rights user U has for object O.

We said there are two parts, we had to define our access control policy and then we have to do enforcement based on what is in that policy. Clearly that policy defines who can access what and things like that. That kind of information, we going to abstract that in a data structure that is called an Access Control Matrix, or an ACM.

If it's a matrix, in this case an access control matrix, that is going to abstract all the state that is relevant for making those access control decisions. Well matrixes have rows and columns. To define a matrix you have to say what rows it has, what kind of columns it has, what does it store in each element, or each cell of the matrix. In an accesss control matrix, rows are defined by the users or subjects that we have in the group. So rows actually correspond to the sources of requests or the subjects or the users. So this access control matrix, if Alice is a user in the system, there is going to be a row for Alice.

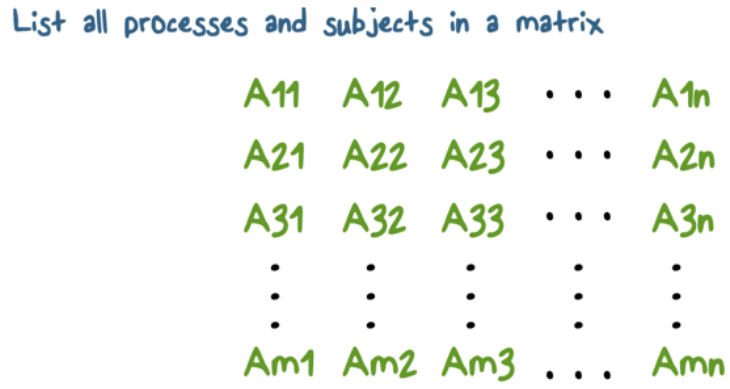
Well the other thing that a matrix has is to define this two dimensional matrix we are talking about is we have to define its columns. Rows correspond to users. Actually columns going to correspond to resources that we have in the system. So each resource that needs to be protected. Remember, we need to know who is allowed to access it or not. So this matrix is sort of telling you for a given user, that's a row, and a given resource or object, what can be done. So the columns are going to correspond to all the resources that need to be protected in the system.

So an access control matrix, since I'm talking about rows and columns and it's two dimensions.

- **ACM defines who can access what**
  - ACM [U,O] define what access rights user U has for object O.

Any particular entry, or cell, in this matrix can be defined by what we have here, [U, O], with the row that is corresponds to user U and the column for object O. You start with a user, continue until you get to the object that we are talking about. So that element of the matrix or cell of the matrix, is actually going to define what kind of access rights user U has for object O. So if this was a file, and the access rights can be read/write, execute or whatever subset of those access rights then we're saying the entry in the matrix, if you look at the row for user U and object O, it's going to say if you can read this object O. R is present in that entry. That means the user can read object O's matrix. Access control matrix is based on users we have in the system, resources we have in the system, and the state actually it captures is who had what kind of access for the resources of the system. That's what each entry of the matrix is going to answer a question for a given user and a given object or resource.

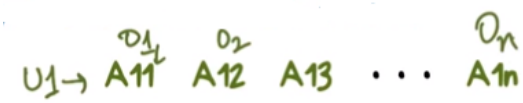
Implementing Access Control



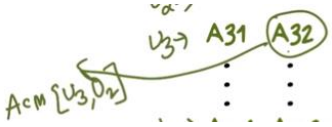
So let's look at this matrix a little bit more.

This is a matrix, the size here is m x n. That's because we have m rows from 1, 2, 3 all the way going up to m. And we have n columns. So if you look at 1 2 all the way up to n.

The first row we know corresponds to the user 1 that we have. Similarly, the second row is going to correspond to user 2, user 3, and the last one is going to be for user m.

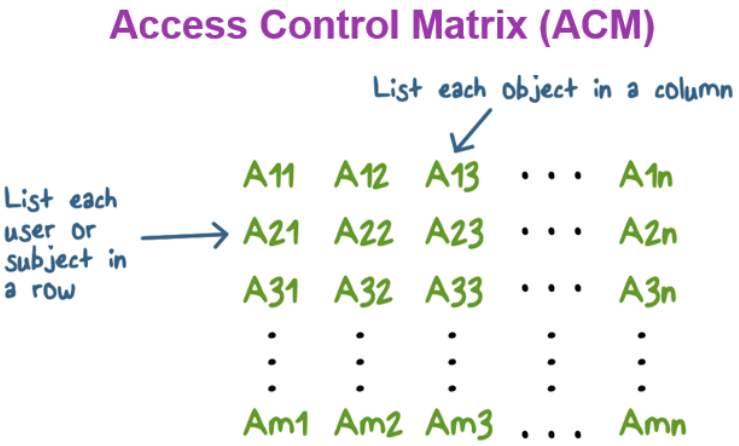


Similarly if we look at the columns, remember these were for objects or resources. So this is for Object 1, this is for Object 2, and all the way to Object n.



So we are talking about ACM U, O. So that's going to be some entry like this. So if you take a look at it, this is are ACM 3, because it's user 3 that we are talking about and object 2. So this entry A32, which is basically going to describe how user three, or U3, is allowed to access object O2. If it happens to a file, it could be read access or write access or execute access. Things like that. So this is what we just discussed for the case where there m users, n objects.

So this is an m by n matrix, a row here corresponds to user, a column here responds to object.



So again if you focus on a particular entry in this one, remember, this is what axis does user 3 have for object 3, A33 is. So this is going to be a subset of the axis right. How this object can be accessed. Object O3 for example here, user 3 we have here. So depending on what kind of object this is, so we've been talking about file objects that contain data which can be read or written. So, the access rights for a file may be read write and execute. So this can be some subset. This could be nothing. That means user 3 could not access object O3 in any way. If it just has read then we know that reads are allowed. User 3 cannot write or execute a file that is object 3. If it read write execute then of course user 3 is allowed to read or write or execute this file that corresponds to object O3.

Data Confidentiality Quiz

Select the best answer to complete this sentence:

A file is created by a certain user who becomes its owner. The owner can choose to provide access to this file to other users. If file data confidentiality is desired, the owner should control who has...

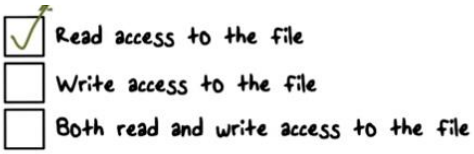
- ☐ Read access to the file
- ☐ Write access to the file
- ☐ Both read and write access to the file

The file is created by a certain user. So the file belongs to user Alice, let's say. She then becomes the owner of this file. And let's say here she can choose to decide who should have access to this file, and in what manner. So if confidentiality is what we are concerned about, then what kind of access should be controlled for this file that contains this confidential data? Okay, so that's the quiz about. Remember, entries in an access controlled matrix cell in a matrix ACM that we just defined, are some subset of access rights. So here is the question of saying, if confidentiality is important what access rights should you focus on? What should be present in that entry in an Access Control Metrics or what right should not not be present in it? So think about the 3 options, and so let the one that you think makes the most sense.

We know that confidentiality is all about disclosure of sensitive data, which when somebody being able to observe or read the data, not about writing it, that is integrity. So if you just concerned about confidentiality or disclosure of this data, what we really need to do is control read access to the files.

Confidentiality is all about disclosure of sensitive data.

Confidentiality is about disclosure and writing is about integrity.



So this is the answer that is correct in this case is that read access has to be controlled.

Determining Access Quiz

Select the best answer to the question:

The access control policy in a system can either define positive access for a certain subject or can specify that the subject be denied access. Consider a case where subject Alice belongs to a group All-Students. The system specifies that members of the group All-Students be able to read file foo but Alice is denied access for it. In such a case, what should the system do?

- ☐ Alice has access because she is member of All-Students so she must be allowed to read foo
- ☐ Negative access should take precedence and Alice's request must be denied

So the scenario here is kind of little interesting one. There is a particular user. The user does belong. User is Alice, she does belong to this group. The system specifies that members of this group can read file foo so that's a positive access right. Think of that as "+r" says in the column it's going to be for file foo, the row is going to be for this subject All-Students we're talking about. And if you look at, in the entry in the access control matrix corresponding to this subject and to this object or file. Then it's going to say, there's. They're allowed to read it.

But Alice is denied access. Maybe Alice is no longer a student, a member of this group that we are talking about. So if you look at Alice's entry for this file foo, okay, so the role would be Alice's role in the access control metrics, and the column is foo. If you go in there, then access is denied. So we have a negative read that says Alice should not be allowed to read this part.

We're exploring a couple of different questions here or ideas here. We're exploring the idea of a group and users being part of a group. The idea of a positive access right as well as a negative access right. And actually the important thing that this question is exploring is what happens when you run into a situation where you kind of seems like you have a conflict? Where Alice is a member of this group and the group has access. But then if you look at Alice herself, she is denied access to this file. So what does enforcement do in this case? That's the question you are exploring in this quiz, and there are two options.

- ☐ Alice has access because she is member of All-Students so she must be allowed to read foo
- ☑ Negative access should take precedence and Alice's request must be denied

When you run into the presence, in the access control metrics, of conflicting access rights, denial is the wise thing to do. Because the policy does say that Alice should not have access. That takes precedence over the fact that she happens to be a member of some group that does have access.

Discretionary Access Control Quiz

In discretionary access control, access to a resource is at the discretion of its owner. Let us assume owner Alice of file foo can choose to grant read access to foo to another user Bob but can prevent Bob from propagating this access right to others. Does this ensure that a third user, Charlie, can never read the data from foo?

- ☐ Yes, Charlie is not granted access so cannot read
- ☐ No, there may be another way for Charlie to access the data from foo

Define something called discretionary access control, it's called DAC. In these kind of policies, who has access to a resource is at the discretion of the owner of that resource.

So Alice owns a file, it's at her discretion whether she's going to allow Bob to read it or write it, or Bob is going to be prevented from accessing it. Furthermore, she can actually say whether Bob can propagate the access to somebody else, access he gets from Alice.

So in this question we are saying, Alice actually is going to grant read access to this file foo, to Bob, but she's not allowing Bob to propagate this access to anybody else. And that is discretionary access control model is basically an Access Control Matrix that we have. She is going to put a read access in the cell or the entry that we have for Bob and foo, okay? So the column for foo and the row for Bob, if you look at ACM, Bob, foo there's going to be read access rights there. So the question is asking, does this ensure that a third user, Charlie, will not be able to read the sensitive data that is in this file foo, or is there some way for Charlie to gain access to this data?

First are the says, does this ensure that can never read the data? So the first one says yes, Charlie does not have access to read this file foo. So Charlie made a direct request to the system, saying, I want to read this file. The system is going to check the access control matrix information that we have. It's not going to find the information for Charlie, so it's going to deny. So Charlie is not going to be able to gain access to the file directly.

Bob can read the data and create a file that he owns. He can copy the data to the new file and give Charlie access.

But things are a little bit more interesting. So what can happen here is that Bob is able to read this file. So what can Bob do here, is read this sensitive data from file foo, and Bob can create a new file that he owns. And he can then tow [copy] the read data into this new file that is owned by him. So the data is now being moved from file foo into a new file that is owned by Bob. And that's possible because Bob could read the data into some buffering memory. And then Bob is writing that buffer into this new file that he just created. The new file that Bob has created is owned by Bob and he, of course, can give permission to read that new file to Charlie. And this new file contains the same data that was there in Alice's file, foo.

- ☐ Yes, Charlie is not granted access so cannot read
- ☑ No, there may be another way for Charlie to access the data from foo

So the answer here actually is going to be the second one. It does not ensure that Charlie will never be able to read the data. This is called information flow problem. Can the information that you shared with someone else flow to another user? And discretionary access control, unfortunately there is no way to prevent that kind of information flow. And so we can't make this guarantee. The only guarantee we can make is that Charlie will not be able to directly read the file foo.

Implementing Access Control

- **Access control matrix is large**
- How do we represent it in the system?
  - Column for object Oi is [(ui1, rights1), (ui2, rights2),…]
    - Called **access control list or ACL**
    - Associated with each resource
  - For user ui, a row in the matrix is [(oi1, rights1), (oi2,rights2,….].
    - Called a capability-list or C-list.
    - Such a C-list stored for each user

How do we implement access control using the access control matrix? This is what we want to discuss.

So first of all the access control matrix is going to be pretty large. There could be tons of resources, depending on what kind of system you have. If it's a large shared server there could be lots of users.

So it's a large matrix, but we know that the matrix is going to be fairly sparse. So sparse matrix here really means that many of the entries in the matrix are going to be null because most users not going to have access for a given resource. Resource is owned by somebody. He or she's going to have access to it and maybe they choose to share it with a few other people.

So the question is how should we represent this matrix in the system? Actually sparse matrices represented in one of two ways. You can represent them either sort of column measure or row measure and that's something we can do here in this case.

So we can focus on each column of the matrix, and if most of the entries are null we can just ignore those and we can make a list that says, well for object Oi, user ui1, this user's access rights are this value. Okay, so we have these entries, so when I said matrices, when they're sparse, they can be implemented through link list for example. So this could be a list of, and it's going to have a small number of entries because most users don't have any access for object Oi.

So if a given user has access, what kind of access we can have, sort of this pair, and then the user id, and access rights, that makes up the first node in our list. Next user is going to be ui2, who has some other set of rights, and so on. So we can just go down the column and see what users actually have known null access rights. And build these entries that become nodes in a link list or something like that. So we can organize this information either by a resource or an object. Here we'll have to keep track of what users have access to this object, and what kind of access they have for this object. So that's one way to do it.

- Column for object Oi is [(ui1, rights1), (ui2, rights2),…]
  - Called **access control list or ACL**
  - Associated with each resource

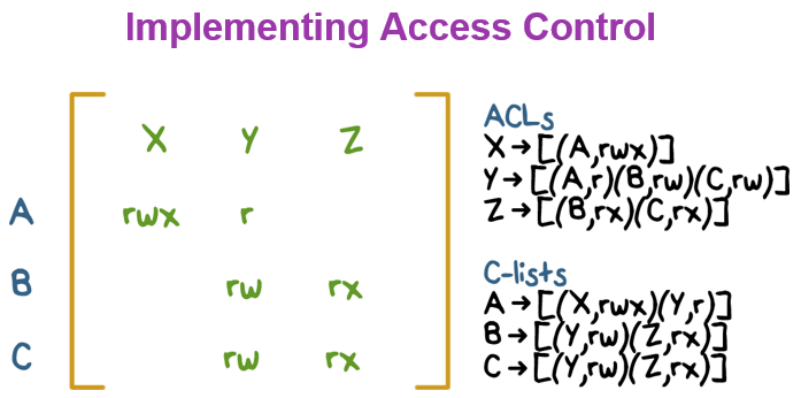
When you do that, this list I'm talking about, or the data structure that we're talking about which is this collection, this linked list kind of a data structure. That's a collection of these entries, what user has what access rights. This is called an access control list or an ACL. So if you focus on a resource or object Oi and grab the information who has access to this resource and what kind of access, that defines an access control list or ACL. To keep in mind is that ACLs are for resources we have in the system. So ACL is always associated with an object or a resource.

There's another way to think about how we can implement it, and rather than focusing on columns, we can focus on rows. And we know that rows correspond to users. So we can go down horizontally, you know, across a row and say, what would we see. If it's a sparse matrix, this user ui is not going to have access for most of the objects. But he or she will have access to some objects. So the first object we run into for which ui has access is this object oi1. So the list we are going to generate here, which is going to say for this given user ui, we do have access and these are the access rights for object oi1. The next object for which we have access, or this user ui has access is oi2 and those access rights are defined by rights2 and so on.

- For user ui, a row in the matrix is [(oi1, rights1), (oi2,rights2,….].
  - Called a capability-list or C-list.
  - Such a C-list stored for each user

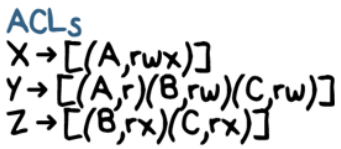
So basically we are ignoring the null entries in the row, and the entries that are not null are for certain objects. So these are those objects, and what kind of access for each of those objects, that's what we capture in this list. Well, this kind of a list is called a capability-list, in some sense what a user is capable of. A user is capable of accessing oi1 in this manner, oi2 in that manner, so it's called a capability-list that is for a given user ui.

Remember, ACLs are for an object, capability-lists are for a user. Again, C-list is going to be associated with each user we have in the system. Remember, we have a row in the access control matrix for each user, and each row really is a C-list. It's a list representation of what's in that row. So, C-list are for users. ACLs are for objects.



The example here says we have a sort of system that has three users. The users are A, B, C, because rows correspond to users. And we have three resources as well, we calling them X, Y and Z. These are three objects of the resources we have. And if we look at the matrix says well A can read write execute. Object X can only read object Y and has no access to object Z. B can read write Y and can read execute Z and things like that. So this matrix actually tells you who can do what given the three users and the three sources we have.

So if you want to implement the information that's in this access control matrix. Well using ACLs then we look at what's at the top here. So what would ACLs or ACLs correspond to object. So the first object is X, first go on you find user A who has read write execute permission. So that's an entry in the list that we're going to, actually in this case the list is only going to have one entry because B has no access, C has no access, those are null entries that we don't need to represent in this list. So the ACL for X it's just going to be this one entry which says, user A can read, write, and execute this object.

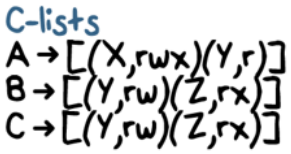


Still talking with ACLs, if you look at Y, the second resource we have here. Actually, here we're going to have three entries in the ACL, one for each user. A is going to be able to read it, B can read, write it. C can read write it, too.

Similarly for the last resource, we're going to have two entries. A has no access, but B and C do, so we can read and execute it and she can read and execute it as well.

So this information that we have in the access control matrix gets represented with three access control lists. One for each object. This is for X, for Y, and for Z.

Now another way to represent this very same information is to using capability lists or C-lists. So remember C-lists are for user. So we have to have a C-list for user A, one for user B, and one for user C.

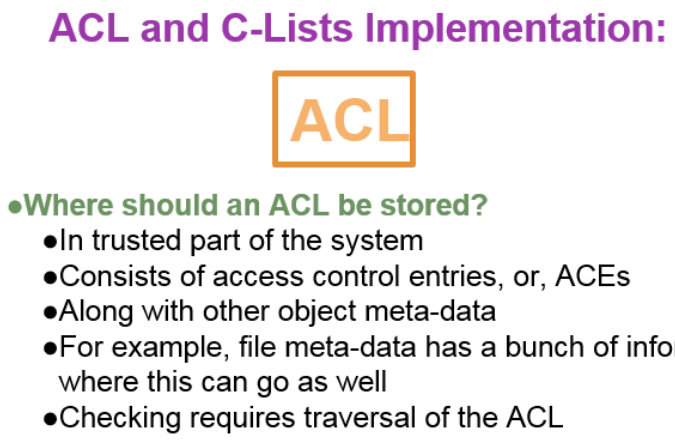


If you look at A, it has access to both X and Y. So the C-list would have two entries. If user A has access for resource X and the access rights are read write execute. Similarly user A can read resource Y. So this is the C-list of user A.

Similarly C-list for user B is going to be no access for resource X, so nothing for X. But for Y we're going to have read write access and similarly for Z we're going to have read execute access. So this C-list is going to have two entries.

And the C-list is going to look the same for user C because B and C have same kind of access. The given set of resources that we have here.

ACLs, you go vertical per resource. C-list, you go horizontal or row-wise per user, so we are talking about how you implement an Access Control Matrix. Well, one way is to define or have this list that we have, and we'll talk about where these lists get stored or maintained, but the ACL or ACM information gets captured either in ACLs or in C-lists.



Remember an ACL is per object or per resource? It tells us user ID, what access that user has, and the next user ID and what access that user has, and some number of entries like that. And by the way, access control entries, or ACEs is what they're called.

So where should ACL for an object be stored? So first of all this has to be stored in the trusted part of the system. It has to be stored in the operating system of the trusted computing base because it actually determines who can access a resource that needs to be protected. If it's not in the trusted part, then some untrusted code or application potentially can change it but where exactly does it go in that system.

So we know that it is a list of these access control entries. And this is for a given object or a resource.

- Consists of access control entries, or, ACEs
- Along with other object meta-data

Such a list exists. And ACL exists for each resource. So we have the resource R is going to have an ACL R.

One natural place for us to store this ACL, is where other information about the resource or the object is stored. So the other information typically what we call is meta-data about that resource. So if the resource was a file for example meta-data might say the size of the file, where on disk it may be stored, who the owner is, and things like that. We can store the ACL along with that other meta-data that you have. So we're going to do an example for the Linux system, Unix systems, how file system access control is implemented. And we're going to see that ACL information actually gets stored same place where meta-data about a file is stored in the operating system.

- Along with other object meta-data

We said meta-data has bunch of other information where ACL is going to be stored as well, so then the question is, how do you use this information?

- For example, file meta-data has a bunch of information where this can go as well
- Checking requires traversal of the ACL

So remember for an object resource if somebody's going to use it, the object or resource has to be activated. So it's ready for use. The meta-data has to be acquired, something has to be set up in the operating system. And once that is done, then the request is going to come. So at that time we have to perform a check, access control check to decide if the request should be granted.

So how do we perform that access control check? Let's say it's coming from Alice. The request source is, we know is Alice's UID, and the request is for file foo. We're going to go to the meta-data for file foo, where we're also going to find ACL for file foo. And once we have that ACL, we basically have to traverse it, looking for an access control entry for Alice. And then see if the access rights in that ACE grant permission to this object to Alice in the manners of what kind of request is being made. So for example, it's read of a file, and does the read access write exist in the ACL or the ACE that we have in the ACL for user Alice.

So traversal basically says when a request comes from a given user, we have to go down the list to see if an ACE exists for the source of the request. And if it does, does it include permissions consistent with the nature of the request. And if that's the case, then we can go on to access. If you can't find such an ACE or the access right doesn't exist, so there is an access ACE that says you can read but the request is for write, then also we are going to deny it. So now you understand how ACLs can be used.

ACL and C-Lists Implementation:

C-list

- **Where do C-lists go?**
  - A capability is an unforgeable reference/handle for a resource
  - User catalogue of capabilities defines what a certain user can access
  - Can be stored in objects/resources themselves (Hydra)
  - Sharing requires propagation of capabilities

The most common systems actually implement ACLs. Operating systems have, in researches I have studied, C-lists as well and we'll see there's some nice properties they offer.

So how would you implement C-lists? So first of all, where would a C-list go? Where would it be stored? Remember, C-lists are per user. C-lists we get from each row of the access control matrix, so it's very user. So the natural place for it is not going to be where we kept ACLs.

- A capability is an unforgeable reference/handle for a resource

The other question one can ask is what exactly is a capability? A capability, remember, says you are capable of accessing that resource. You've been given permission to access that resource. So typically a capability implementation details differ, but is really a handle for a resource. A reference allows you to find or locate the resource and be able to use it. So if you can go use the resource, you shouldn't be able to forge a capability for that resource. Capabilities are sort of identifiers you can think of them or references or handles. But one property the system has to guarantee, is that these capabilities or handle that we're talking about are unforgeable.

- User catalogue of capabilities defines what a certain user can access

So think about the user has capabilities for a bunch of resources. List of those capabilities is what defines a C-list for that given user. The way where C-list is going to go, we have to have a catalog for each user that we have in the system. Similar to ACLs for each resource or object that we had before, here we have to have a catalog of capabilities, and system has to store the catalog of capabilities for each user that we have in the system.

- Can be stored in objects/resources themselves (Hydra)

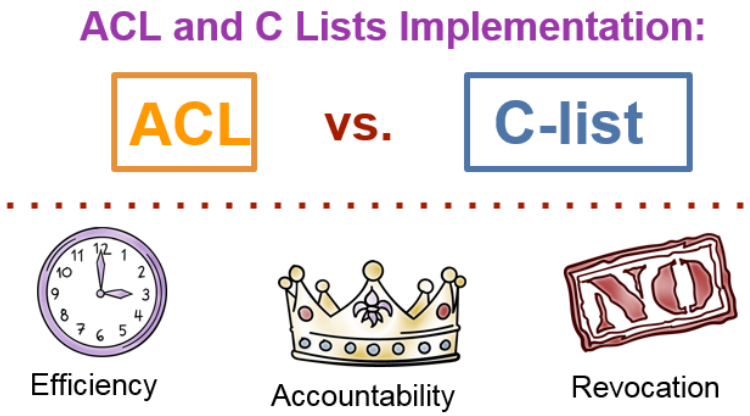
The main capability systems that have been explored and one of the interesting ones was a system called Hydra. It was sort of a research project done at Carnegie Mellon. This actually, you could have C-list stored in objects themselves. So everything was an object. It was an object based system. And objects stored a C-list. So when you came to a certain object, well you found new set of capabilities. And you could then use those to access objects that you couldn't access before. You can access this new objects while you are within this object. And you could take capabilities back to the object it came from sometimes and not other times, and things like that. The basic idea is that capabilities are how you decide, or the system makes the determination that you can access or not. So you, the user has to start with a catalog and as you execute maybe new capabilities come your way. And that was this example we're talking about.

- Sharing requires propagation of capabilities

So one thing we didn't talk about is actually how does sharing happen. The whole idea of who can access what, and giving access to somebody else or the example or the quiz question we've been doing, is how does sharing happen? With ACLs, we didn't talk about it. But someone who has the right to let somebody else access a resource and make a system column have this new user or this user that I'm specifying should be able to read this file foo and what we're going to do is create a new ACE for this

user, if one doesn't already exist, and going to add this new access right to that and then put that ACE in the ACL. So the new user will be able to go and access this file in the future.

Capabilities, what do we have to do? Sharing is going to requite propagation of capabilities. The new user which should be able to access this resource In the future. We have to provide this user with the capability for this resource. So the capability has to be propagated. But remember they shouldn't be able to forge it, shouldn't be able to mint a capability unless it has been passed to them or someone had chosen to share the resource with them. With capabilities sharing is going to happen by propagation of capabilities. I'm going to give you a capability, you're going to find a way for you to gain access to a capability that you can then use to be able to use the resource. Remember, the one difference here is that possession of a capability means you can access the resource. There is no access check required as was necessary with ACLs.



So when there's more than one way of doing something, well of course we have to decide if you're implementing an operating system or something like that, should I go with ACLs or should I use C-lists? I said most operating systems use access control lists, but some were done capability oriented or capability based. So there must be some pros and cons. One must have some nice things that perhaps the other one doesn't offer and then you have to see what is more important to you and based on that you're going to make a decision. Let's look at couple of these possibilities. Metrics that you perhaps can use to decide whether you want to go with ACL or C-list based implementation access control.

The first thing we worry about is efficiency. So in this context, I said the couple of things when we're talking about implementation with ACLs and C-lists, with ACL I said you have to traverse the list. You have to go down the list looking for an Access Control Entry, or ACE, for the user who made the request. Traversing a list is there is some processing overhead required and the compute time it's going to take. Capability list I said it's a handle, shouldn't be able to forge it but once you have it it's been propagated to you. You present it to the system and system knows that by mere possession of this capability, you should be able to access the resource. So when comes to efficiency, C-Lists is faster. We don't need to traverse anything but ACLs have this negative thing were we have to go down the list. So efficiency would say c-lists are better.

But lets look at another thing that's good to have, and that's accountability. So let's say if I ask you who all have access to this sensitive file? With ACL, what you're going to do is you're going to find the ACL, you'll go down the access control list, and you have every user and what kind of access they have for this file. So information about who can access this file is available. All of it is available in one place which is the ACL. So ACLs are actually good when it comes to accountability because we can go look in one place and be able to answer that question. How about C-list? Remember we said C-lists are stored either in objects or catalogs for different users and so on. So to find out who all have the capabilities, who are all the users who hold the capability for resource, I have to look on every user backlog. I have to look into every object that may store this capability, for example, if you want that capability to be to be used from that particular object. So now accountability is going to be hard because as we said we may have to look at all of the catalogs and things like that. So C-lists are not so great when it comes to accountability.

Revocation is essentially you give someone permission to access a resource but in the future you don't need to share the resource with them anymore. So at that point you will revoke their access, remove their access. With ACL, how can we do it? So let's say Alice decides to revoke Bob's access for file foo. Well, it's easy. We know that there's an ACL for foo. And Alice makes a call to the system saying remove ops permission. The system locates the ACL, finds the ACE for Bob and in there removes whatever permission that's been revoked. So the ACL revocation is actually easy. They have this desirable positive property because we know how to go turn of the access right in the ACE, and all the ACEs are in one place. How about C-List? How do we revoke with capabilities? Well that's interesting, it's actually hard because we know the capability sits in Bob's catalog, and Alice can't go remove capabilities from Bob's catalog at will. How exactly does she do it? There are interesting ways in which you may be able to do it and things like that. But with C-lists actually revocation is not easy.

So if you look at accountability, of course, C-list look better. If you look at efficiency, C-List look better. But if you look at accountability and revocation, ACLs look better. So there obviously these tradeoffs we have to worry about and because of that I said most operating systems actually choose to go the ACL route but we're going to see how maybe we can get the best of both worlds.

ACE Quiz

Select the best answer:

Alice goes to a movie theater and purchases a ticket for her favorite movie. She is allowed access to the movie because she has the ticket. The ticket is more like a...

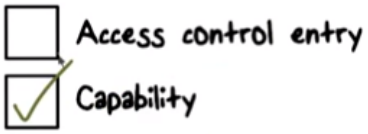
- ☐ Access control entry
- ☐ Capability

So here we're asking, is the access control that's happening. Is it kind of based on access control lists or access control entry? Or is it being based on a capability?

A scenario is Alice going to a theater purchasing a ticket, presenting the ticket to whoever is doing admission control at the theater. So the ticket, is it more like an access control entry, or is it more like a capability. That's the question.

The ticket really is a capability. The possession of that ticket allows Alice to go in the theater, b e able to access the movie in this case. The reason it's a capability is, it's not an Access Control Entry. It's that, the person who's checking the ticket doesn't even need to know who Alice is actually. The mere possession of the ticket. Remember that sounds like what we were saying about capabilities. Mere possession of a capability actually allows you to gain access to the resource. In this case, the ticket really is a capability. We don't even know the user ID of Alice or the person who's doing admission control doesn't even know who Alice is so it's not an ACE. Because an ACE remember requires the UID and what kind of permissions they have. So this quiz the ticket is a capability.

The ticket taker does not need to know who holds the ticket. Mere possession of the ticket is enough for access.



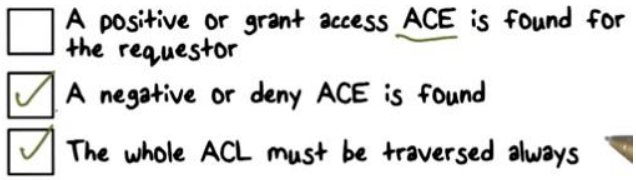
ACE Access Quiz

Select the best answer:

Some operating systems (e.g., Windows) include deny or negative access rights. In this case, an access check procedure can terminate as soon as...

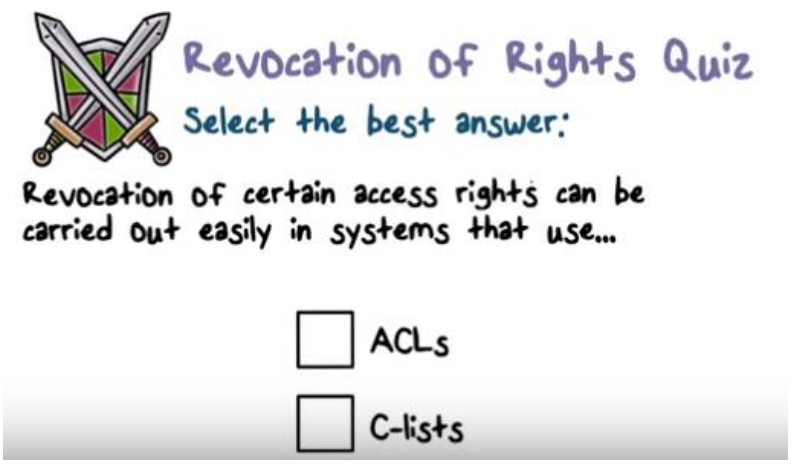
- ☐ A positive or grant access ACE is found for the requestor
- ☐ A negative or deny ACE is found
- ☐ The whole ACL must be traversed always

In this case, how does the ACL traversal happen? That's what the question is about. You have a couple of different options. Remember in this case, we want to grant access when there's a positive access right, and no negative access, okay? Because negative takes precedence over positive, that's what we have discussed. So for someone, there's a number of cases you can think of. If there's no positive and no negative access, then access should not be granted, because there's no access. If there's positive and no negative access then access should be granted. If there's both positive and negative, the negative takes precedence and negative should not be granted. So in this case, how does this get implemented is what the question is about.



The first one says, we are traversing the ACL and we run into an ACE that says, this user does have access to this resource. You can stop right there. So this talking about termination of traversal that we are doing. Saying, you stop right there and grant access to the user who's making the request. Oh, this is not correct, because we said, there is a possibility that there is another ACE that contains a negative access. So we said all students has positive, Alice had negative, so then we have to go to Alice's ACE as well. So we can't stop at the first one, positive one that we run into. So this one, is not a correct answer. The next one, a negative or deny ACE is found, you actually can stop right there because negative takes precedence. If there's a negative ACE, then maybe a positive somebody else in the list, it doesn't matter, okay? Because this is going to take precedence anyway. Or, sometimes we may have to traverse unless use or implement a list in a smart fashion, that actually Windows does do that. But and it does that by putting negative ACE's in the front of the list. But if you don't do that

optimization or that smart thing that I just mentioned, then you have to actually traverse the entire list to make sure that there is no negative or deny access ACE anywhere in the list.



We sort of compared ACLs with C-lists, and revocation was one of those things. Remember, we want to remove some user's permissions. So, it is easier to do this with ACLs or C-lists? That is the question.

It is easier with ACLs because you traverse and find the ACE for the user for whom you want to remove the access right, and update that ACE or actually remove that ACE from the list if the user has no more access rights left. So we know what resource that we are talking about, or which resource access is being revoked. We have the ACL for that resource, because all the ACEs are in one place. We find this user's ACE, and we take care of it.

C-lists, we said that capabilities are in the user's catalog, and things like that. We need to get the remove capabilities from certain places, which is harder to do. So revocation is easier to do with ACLs.

Access Control Implementation

How is Access Control Implemented in Unix-like Systems?

- In Unix, **each resource looks like a file**.
- Each file has an owner (UID) and access is possible for owner, group and everyone (world).
- **Permissions are read, write and execute.**
- Original ACL implementation had a compact fixed size representation (9 bits)
- **Now full ACL support is available in many variants** (Linux, BSD, MacOS,..)
- Few other things (sticky bit, setuid,…)

Now we are going to talk about how actually Access Control is implemented in real operating systems. We actually going to talk about Unix-like systems, but there is similar things done in other operating systems. Let's focus on Unix-like systems.

And so we have to think about users, resources, access permissions and things like that for us to understand how access controlled is handled by this system.

So in Unix, actually every resource for which access needs to be controlled looks like a file.

- In Unix, **each resource looks like a file**.

So in your next like system we also know that users have UID. Unique identifiers for the

- Each file has an owner (UID) and access is possible for owner, group and everyone (world).

users that we have in the system. Users can also be groups and certain users could be members of a group.

There is a special group. Every user in system that's the world group. And we look at access control for a file, we're going to say, well, who is the owner? What kind of access does the owner have? Is there some group that access to this file? What kind of access that is, and what about everybody, people who are not owner or members of this group?

- **Permissions are read, write and execute.**

Since resources of files, they can be read, written or executed, these are the three access rights. So at this point, we know, we think about your access control matrix instruction, rows would correspond to each UID and each group ID that we have, GID and a special group world that we have. Column is going to correspond to each file that we have in the system and if you look at an entry in the access control matrix it's going to be a subset of read write execute.

- Original ACL implementation had a compact fixed size representation (9 bits)

Unix system, the original ACL implementation was actually, had a compact fixed size implementation. Remember, ACLs are ACEs, so that's for each user, so the owner is one of the users, then the group would be the next one, and then everybody is this special world group that we are talking about. So actually had only three maximum of three ACEs. Actually had three possible ACEs and ACL it wasn't implemented as a link list. It wasn't implemented. It was implemented as a bit mask. In particular, we need nine bits. So the nine bits encoded read, write, execute for owner. Let's say the bit is on for read for a given write, then the user has that write. So the user can read it. This is how the compact fixed sized representation, perhaps in those days seven memory was important, so they implemented ACLs using this essentially nine bits. Little bit more complicated, but that's the main idea. A few things, few extra bits we'll talk about.

- **Now full ACL support is available in many variants** (Linux, BSD, MacOS,..)

Now this does have its limitations, for example, you can only have three subjects or principle whom access can be either just the owner or just one group or everybody. If we had 10 different users, you want to get information to six or three or whatever it is, we can't have an arbitrary size access control list with this representation we had here. So operating systems actually now provide implementations of full ACLs, and it's available in many operating systems, Linux, MacOS, BSD, and so on.

- Few other things (sticky bit, setuid,…)

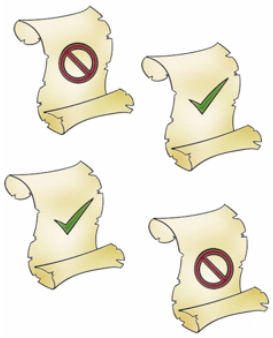
I did say that I'm oversimplifying, and there are a few other things. So there are a few other bits that can go with this other information that we are talking about. In particular, there's an interesting thing called setuid, and maybe we want to talk about this a little bit.

Think about a game program that needs to update a file that stores the scores of different people who have played this game. So when these people are playing the game, of course, we want to be able to update that file. When your score increases you want to update your score that's in there. So, these users while playing the game, and they play the game, need to have write access to that file. When they're not playing the game, we don't want them to have write access because then they can go change the score without playing the game, so we don't want that. So how can we solve this problem?

These systems I'm talking about, this problem is solved by having something called a setuid bit. So the way to understand this is that we have Alice and Bob, and Bob owns the game file, and Alice wants to play the game. So the game file is actually owned by Bob, and Bob is going to give permission to Alice to execute, being able to execute this file. By the way, the score file is also, Bob is going to own the score, and is not going to grant write access to Alice because Alice should not be able to update that file unless she's playing the game. So you can think about the score file, Alice can read it, her score and somebody else's score, how people are doing, but she can't write it. Bob can rewrite it because Bob owns the game file that we have here. He's some special user, or someone who is running this program that we have. People who want to play the game.

So, setuid bit is set on the game file, the executable file that we have. And we said Alice executes this file when she wants to play the game. So setuid bit, when it's set on executable file, actually does an interesting thing. When Alice executes this file, the User ID temporarily changes it from Alice to Bob. So it says when a setuid file is executed, the User ID, effective User ID, at that point is not the user who actually executed the file, but is owner of this file. In this case, the owner of the file is Bob. So during the execution of the game, although Alice is playing the game, during the execution of the game, the temporary or effective UID is Bob, and because of that, even though Alice is playing the game, she is able to update the file. File is updated back that is in this game file. And while we're executing the game file, as we said, the effective User ID becomes Bob. Alice launched this program, but the program is owned by somebody else. This change in UID is possible when the executable file that Alice executes, the game file here, has a setuid bit set. So a setuid bit is used to change the UID temporarily during the execution of the program on which this bit is set, and I hope I motivated why you may want to do that, in this case, for example, the game file that we are talking about.

Access Control Implementation



How are files used (system calls for accessing files)?

- Create (filename) /* several ways to do it */
- fd = open (filename, mode)
- read (fd, buf, sizeof(buf))
- write (fd, buf, sizeof(buf))
- close(fd)

So we have been talking about implementation of access control and the resources and request for those resources and we said well in Linux based systems, resources look like files. So let's actually see how the resources are used and you know what kind of calls we make for accessing them and what happens during those calls.

So, first of all, of course, resource has to be created. So the process that creates this resource is the owner. That owner can have whatever set of access rights they want to have. In a Unix-like system, you probably know the number of ways in which you can create a file, for example.

So once a file exists and we want to access it, you do what is called open the file. So this is sort of like prepping the file. If it doesn't exist, then file gets created. When you do that, file has a name and mode here says, do you want to read this file or do you want to write it or execute it. What are you opening it for? How do you want to access it? So, once that file exists, the first thing we do is open it. When you open it the operating system returns to you a descriptor which is a small number, and we'll see what the

use of that is, but as a result we'll see that to be able to access or get the data that's in the file, write it into the file, we're going to need the file's descriptor. So the descriptor comes when you open the file, you prep it or get it ready for access. So a system call you make saying, this is the file I want to access, this is how I access it. The operating system prepares the file for accessing, returns you this file descriptor that you then hold on to.

- Create (filename) /* several ways to do it */
- fd = open (filename, mode)
- read (fd, buf, sizeof(buf))
- write (fd, buf, sizeof(buf))
- close(fd)

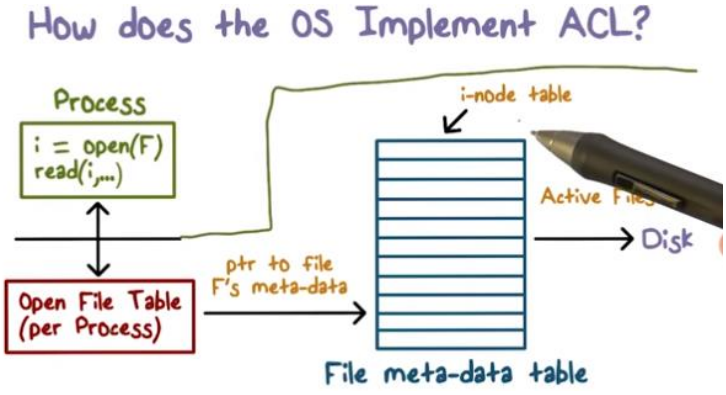
So if you want to read this file in the read call, this is another system call that you have. You have to go to the system. You specify what file using a descriptor. This is what was returned here. It's a small number. So this could be file for example. If you're reading you are to specify where the read data should be put, how much data you are reading. That's the buffer, and the size. Data is going to come from the file. Where is this data? Well, the file we know, unless file point or that points at the next place from where your data is going to come and it is advance that you read. So whatever data follows the file, current file pointer, we are going to pull that data, lays that in the buffer and that's what the recall is going to do.

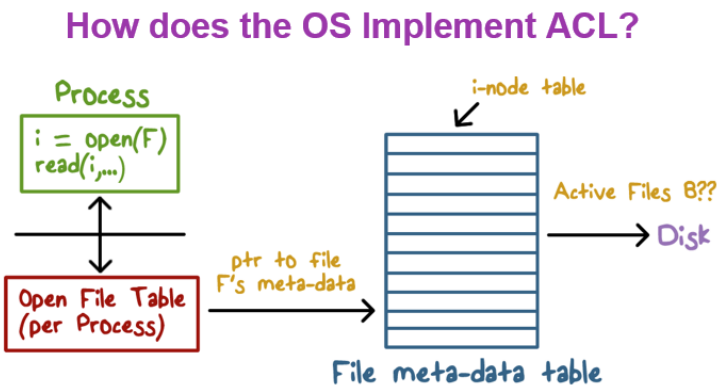
Similarly there's a write call, except that the data flows in the opposite direction, so the data is here then it's going to go into the file. And how much data, that's what we specify in this call. So an open is followed by one or more read or writes.

And when you're done you close the file. And once you close the file, again the input here is the descriptor that says what file that you had open before. Then this file we are basically telling the system is that we're not going to need to access it anymore. So the system can, if I'd set up some data structures, it can essentially free those up at that point.

A resource is a file. Users have UID, we also talked about group ID, and to access a resource, this is what we basically have to go through, file exists, they'll open it, then then they have to say read or write it, and then you close it. If you forget to close it and the process terminates, it's implicitly closed, but this is how a file gets used.

Talk a little bit more about what happens when you open a file, and how does access control factor into it. So, process here is making these calls we're talking about. It first opens this file F, without specifying the mode here, but maybe this is read. This line is going from user to the operating system, system call, this is the boundary between the two. This is untrusted, this is trusted, as we talked about. So, this is all in the operating system.



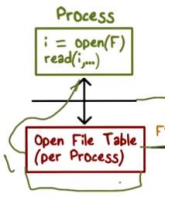


So basically, the file got stored on the disk, and files that are actively being accessed, the operating system keeps track of those information about a particular file is in a file control block. So if you've done the operating system's class, you have some idea of how file systems are implemented. But for each file that we want to access, we you know meta-data that we're talking about for a file you know what's the file pointer, where should we read next, where on disk the file blocks are. Other kind of information. We keep track of that in the file control block.

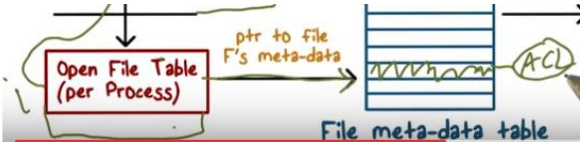
In Unix, well that's called an i-node, and there's a table of these. That's where this i-node table stores the meta-data about the active or currently open files. So this is, here, I'm saying it file meta-data table.

A file that you're going to open has to be active, it it's not active, it's going to get activated. So, there's something interesting happen, when you open a file you tell the filename, the operating system's actually is going to go and say, is this file currently active. If it is active then there is going to be a file control block for it, okay. That's where the meta-data for this file is going to be.

So a process also has another data structure that's called an open file table. Okay this is kept by the operating system. It's per process data structure we're talking about. So in this open file table that we have for this process that is making this open call, basically what we want to do in the open call is that we want to see if the file is active. If it is then there is a meta-data block or file control block for it. In the open file table, there is some set of entries that we have. This is a fixed sized table. We're going to use one of those entries to actually store a pointer to this. And the file descriptor we return is actually nothing but an index into this table.



So i is the index in this table that we're returning. I think it was about access check when do we perform. Actually in Unix-based systems, access control is done when you open the file. At the time the file is opened remember we're going to go in, we're going to find out if the file is currently active or not, we're going to locate its file into a block where the meta-data is. In particular the ACL is here too.



With the ACL is here, then it was nine bits that I was talking about. Depending on the user or group the user is part of. I'm actually going to look at those ACL bits that we're discussing. And if the mode here is read, and you have read information, I do all this. Okay? I'm going to grant you access Mode is right and the right permission is here. I'm going to do all of this and give you write permission and to place data in this file. So access check is going to happen during the open calls execution in the operating system because we locate this information where the ACL information is and we're going to look that up. And if everything checks out we set this up and return this file descriptor.

So the reason we do it at the time it's open is, that when you come to read, you specify a file descriptor. So, it's going to say read i here, for example, that's what you're saying. Well, once I come to the operating system, it's the I'th that remember is an index into a table, that tells me, it's this particular entry in the table. Follow the pointer that it points to, that's where the file meta-data is, and that tells you where the file data is.

I don't do any access check at the time I do a read or write. I just sort of quickly follow these pointers and get to the file. If I'm doing reads then I find the file data to return to copy that into the buffer. If I'm doing a write then I copy the data from the buffer into file cache or something like that. It gets copied. So there's no check done during either read call or a write call. The check was done when we did the open, okay. So the fact that the script was returned to you at that time we did the checking and now you're able to do that. If we opened it for read and you tried to do write, of course. Of course we're going to stop that. Okay. That goes without saying. We don't have to go look for the particular to do the transversal that we are talking about.

Time to Check vs. Time to Use (TOCTOU) Quiz

A time-to-check-time-to-use vulnerability arises when access check is performed separately from when a file is read or written. TOCTOU vulnerability arises when...

- ☐ File permissions change after an open() call completes for the file and before it is closed.
- ☐ The file permission change only when the file is currently not opened by any program

So we want to talk about something called a Time to Check versus a Time to Use problem:

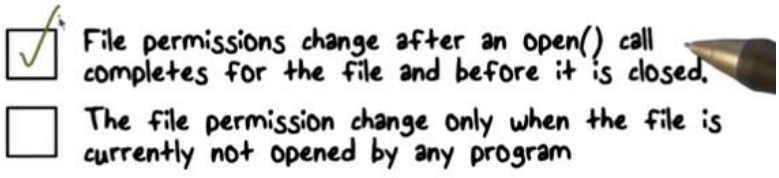
- Time to Check is when you open the file.
- Time to Use is when you either read or write it.

So a time to check, time to use vulnerability arises because access check is performed separately from use. Typically when you say you want to use a resource, you perform the check. But the model that I just showed you with the open call coming first and then read writes happening later on, and you can do as long as the file is in the closed and the process is running. Okay, so it could be a while before you do your read or write after the open was done. So when the two are separated, this vulnerability arises. And based on what time to check versus time to use, think about what these are. I want you to look into these two options that we have. What is the reason for this TOCTOU [Time to Check vs Time to Use] vulnerability that we're talking about?

So, let's look at the two options that we have here. First one is file permissions can change after an open() call – the checking was done when the open was done. That is when we went down to the i node table or the file meta-data table, set up the file open table, return an index, and all the checking got done. Well, at that time, this user, say Alice, was able to read the file. Later on, Bob, who owns the file, turns off read permission for Alice. But Alice's process is still running – Alice's process actually opened the file before Bob was able to revoke permissions, and Alice's process has not closed the file. Will Alice be able to continue to read this file? Yes, because we are not performing any more checks, the checks

were done at the time we opened the file. Okay. So it was separated from the use, so the check was during the open call. Okay, the sequence is as follows: Alice opens the file, and sometime after that Bob comes in and removes permissions, and sometime after that, Alice is trying to read the file. Alice is doing read, not open now, so there are no check happening. So if file permissions change after an open call completes, but before it is closed, Alice has the file open and has not closed it. Because if she closes it, then she has to open it again and check again. And when we check again, Bob's change might be seen by the system.

So if permissions change between the time that we check and the time that we use, well that actually is the reason for this vulnerability, TOCTOU vulnerability that we talked about.



And the second option is the file permissions change only when the file is not currently opened by any program, then this vulnerability will not arise.

Unix File Sharing Quiz

In Unix based systems, a file can be shared by sharing its descriptor.

- ☐ True
- ☐ False

We talked about how file system implementation works in Unix-like systems or Unix based systems. And said open returns a file descriptor which is a small number. And the question is can a process share a file with another process by just sharing that file descriptor?

The answer here is false. You cannot do the sharing by sharing the file descriptor numbers because those descriptors point into a per process table we're talking about. Descriptor five value FD being five means the fifth entry in P1's open file table points to something. Okay, but process P2's open file tables is entirely separate. So fifth entry in there may be pointing to nothing or may be pointing to altogether different file. So if you try to use the descriptor, it won't take you to the right place, in particular the file that process P1 is trying to share it. Well it has the operating system data structure set up to get to it, and these are per process, so the same thing as done for P2, it's not meaningful to share file descriptors in Unix based systems, so the answer here is false.

SetUID Bit Quiz

An executable file F1 has the setuid bit set and is owned by user U1. When user U2 executes F1 (assuming U2 has execute permission for F1), the UID of the process executing F1 is...

- ☐ U1
- ☐ U2

We talked about the SetUID bit and in particular a game file and a game score file and things like that. So this question is about the SetUID bit. In particular executable file F1, so that's a program file has its SetUID bit set. And it's owned by user U1. User U2 executes F1. So we are assuming that U1 who owns F1 has given permission to this file to U2, which user ID is the effective UID when F1 is being executed in this case.

When the setUID bit is set, the UID temporarily changes to the owner of the file that is getting executed on which the said UID which was set. Owner is U1. So although the process that we're talking about was started by U2, during the execution of this file F1 One, user ID is going to change to U1. And it's going to change because F1 has the SetUID bit set. And when the SetUID bit is set, the effective UID becomes whoever the owner of this file F1 is. So in this case, the answer is U1.

Role-Based Access Control (RBAC)

- In enterprise setting, **access may be based on job function or role of a user**
  - Payroll manager, project member etc.
  - Access rights are associated with roles
- **Users authenticate themselves** to the system
- **Users then can activate one or more roles** for themselves

In some systems, what we have is Role-Based Access Control (RBAC). Roles, sort of, is what you do, your function perhaps. That may govern what kind of files you can access. If you are Human Resources versus Payroll or if you are in a different department, the set of files that you can access would depend on what your role in the organization is. So there is Role-Based Access Control, or RBAC that is implemented. It is a little different, so I thought that I would spend just a couple of minutes talking about it.

In Role-Back Access Control, remember rights are for being able to access certain resources, or access rights. Those are associated with roles, they are not associated with users. So the way to think about the way you are going to define your policy access, the access policy is going to say which roles do I have in my system, and for each role, what kind of resources do they need to have access to? So, people in this role can read these files and write to these files, and things like that. So, this part of the policy basically says that access rights are defined for roles, and users that are going to log into the system, or authenticate themselves to the system, can then take on some roles. So, RBAC, the way to think about this is that we don't have direct access rights for users. Users must be activated into one or more roles, and once they assume one or more roles, based on what those roles give them access to, that is what this user, or the process running on behalf of this user, can access.

- In enterprise setting, **access may be based on job function or role of a user**
  - Payroll manager, project member etc.
  - Access rights are associated with roles

So, we are talking about enterprise setting, access may be based on job function or role of a given user. That's where this might make sense. Project manager might have access to all files, developer or people doing QA or something like that may access files in a different way, and so on. Payroll manager, depending on their function, may be able to look at files that have people's salaries, and so on, because they are payroll manager, and things like that. So this is how roles will be defined. Access rights are associated with roles, as we said before.

- **Users authenticate themselves** to the system

And users and authentication happen and we know who the user is, who is logged into the system. And we have to start with authentication, and then be able to activate one or more roles for them.

So, role activation is something else that has to be added to this process. The policy defined here, and sort of, there are two stages of activating a role for a user. And based on that deciding what a user can access.

RBAC Benefits

- Policy **need not be updated when a certain person with a role leaves** the organization
- New employee **should be able to activate the desired role**
- **Revisiting least privilege**
  - User in one role has access to a subset of the files
  - Switch roles to gain access to other resources
- SELinux supports RBAC

Now there are some benefits when we have Role-Based Access Control. What could be these benefits?

First of all, we said that our policy defines what kind roles have what kind of access to the resources in the system. So the policy does not need to change when let's say a certain person leaves the organization. Policy is associated with roles, it is not associated with users. So, users coming or going, they don't require changes to the policy the organization has.

When a new employee comes, basically we think about what role is appropriate for them. And as soon as we decide the role based on their function, and what resources they should have access to automatically happens. Because that role along with it has a set of access rights to various resources, and that, as I said, happens automatically as soon as we decide what role this new employee can take.

An interesting thing, least privilege remember is one of the design principles that we had. We said that you should always execute with the smallest number of privileges or access rights that you need to do what is being done at that time. And it is really a damaged 10 minute thing [???] – if something goes wrong, you don't negatively impact the resources that you had no business having access to at that point.

So, how does RBAC help you with that? Well, actually is does because users can start in one role, and access a subset of the files that are only available to that role. The user can then switch roles and go after a different set of roles, a different set of files that are associated with the new role. If you don't have RBAC, the user has a user ID and has access to everything he or she can ever access. Roles sort of

give you the ability to control thing [???], if you are in a certain role at a given time, only the resources that are needed for that role should be available at that point, and we can do that with RBAC.

You know being able to implement least privilege is a good thing that is a design principle for systems that we want to trust. So RBAC actually enables that. So there are actually systems that implement RBAC – SELinux, we are going to come back to it, is Security Enhanced Linux, actually supports RBAC, and there are others as well.

- SELinux supports RBAC

RBAC Benefits Quiz

In systems that do not support RBAC but allow user groups to be defined, benefits of RBAC can be realized with groups.

☐ True

☐ False

In a system, our systems that don't support RBAC but allow user groups to be defined, benefits of RBAC can be realized with groups. The question is can the benefits of RBAC be realized with groups? True that groups are basically give you the same benefits that RBAC gives you or is it false?

And the way to sort of think about this role is sort of more based on a function or job, which is closer to what kind of resources you need for that. If you're a payroll manager you need access to payroll files, for example. That's what you're talking about. So roles are sort of characterized more by function and the resources that go along with that function. Groups, on the other hand, are basically what set of users or what UIDs or what subjects have some sort of similarity so they can be grouped together.

Okay, so roles are sort of more resource-related versus groups are more subject or user-related. So that's sort of the way to think about the two. And then the question is can we get the benefits. So depending on how you actually implement groups, role activation could be like activating a group for yourself. And deactivation is leaving the group and having permissions associated with the group. So only when you activate your members in the group you able to access it. So in theory, the answer is that you could actually do the kind of things you can do with roles with groups as well.

But there's sort of this, basically the difference is in how the two come about that we talked about. But the way, actually, groups are typically implemented in Unix-based systems, actually they don't work the way that I was talking about. You don't join a group and then leave it and things like that. If you used how we implement groups and define access based on those groups, if you want to say I want to go and basically use that to get RBAC, you're not going to be able to get that in systems that we have.

So there is a difference between the two, and I said the difference is somewhat subtle. Possible to sort of think, subject-focused then centric, then you are sort of in the groups. Or object or resource-centric, then you are with roles based on the job function. But if we look at how we implement groups, and what kind of functionality that you want to have with roles, I would go with the false answer here to this question.

☐ True

☑ False

Access Control Policy Quiz

Fail-safe defaults implies that when an access control policy is silent about access to a certain user U ...

- ☐ Access must be denied when U makes a request
- ☐ Access can be granted because it is not explicitly denied

We talked about another design principle, which is fail-safe defaults. What it implies is that when access control policy is silent, if it fails to be explicit about how someone has access or not, what should the system do in that case?

What is the fail-safe default when explicit access is not defined. So these are two options and if you are following fail-safe default, which one would you pick, is this question.

Well actually fail-safe says you should deny it. If access is not provided explicitly, then the default is deny. So the only way for someone to gain access to a resource is that it is guaranteed to them. If we're silent about access being granted, then basically we assume it should not be, so the default is deny. And the first option is the right one.

- ☑ Access must be denied when U makes a request
- ☐ Access can be granted because it is not explicitly denied

Access Control Lesson Summary

- Fundamental requirement when **resources need to be protected**
- An **access control matrix** captures who can access what and the manner in which it can be done
- **ACLs and C-lists** are ways for implementing access control
- Getting **access control policy right is challenging**

We saw that Access Control is a fundamental requirement for computer systems that protect resources. We started with the Access Control Matrix Abstraction and how we can implement using Access Control Lists and capabilities. We also explored how operating systems implement Access Control