

IIT Madras

ONLINE DEGREE

Modern Application Development – 1
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Access Control

Hello everyone and welcome to this course on Modern Application Development.

(Refer Slide Time: 00:17)



Hello, everyone. In this set of lectures we are going to look at the topic of security. Security refers to multiple things. There is access control, the mechanisms that are used to implement it on the web, various issues related to session management, HTTPS, which is used in order to provide secure connections, and finally the topic of logs and analysis, all of which we will be looking at later.

(Refer Slide Time: 00:41)

Access Control



So, let us first start by looking at the topic of access control. What is it, how does it need to be implemented?

(Refer Slide Time: 00:50)

What is access control?



- Access: being able to read/write/modify information
- Not all parts of application for public access
 - Personal, Financial, Company, Grades, ...
- Types of access:
 - read-only
 - read-write (CRUD)
 - modify but not create
 - ...

So, the first thing, what is access control? And access in the context of a computer, generally means being able to read or write or modify certain information. Now, not all parts of an application are typically meant for public access. So, even if you are using, let us say an e-commerce application, definitely there would be some information, like even your actual, your phone number, your email address, your physical address, which you do not want to be available to everyone who visits the site. Obviously, the e-commerce vendor needs to have that

information so that they can contact you and they can ship the, whatever it is that you are buying to you.

But a random visitor who is coming along and seeing this does not need to know your phone number or your physical address. And in fact, that can actually lead to problems. Similarly, financial information. If you have a bank account and somebody else also has an account with the same bank there is no reason why you should be able to see each other's accounts. Inside a company there will be compartmentalization, different teams need to have access to different types of information. In a college or university the students would need to have access to their own grades but not necessarily those of their classmates or others in the college and so on.

So, that, controlling that access to the information is essentially what access control is all about. And there are different kinds of access. There is read only access, and as we saw with the examples over here, even in read only you need to have access control. So, it is not just a question of who can go and modify your grade inside the records. I could make a case that I do not want somebody else looking at some, a person's grades without having a good reason to either.

Now, read only of course, is there. But other than that, you could also have read-write which is the entire sort of trend, create read, update read kind of operation. In some cases, you might have options. In fact you might be able to give more fine grained control. You might say that look there is a page that I have already created. And a particular student perhaps is allowed to go and modify certain parts of it, or certain information on it. But they are not allowed to create a new page of their own and they are not allowed to completely delete this page either. So, access control can be sort of seen at various different levels in various different contexts.

(Refer Slide Time: 03:24)



Examples

- Linux files:
 - owner, group: access your own files, cannot modify (or even read?) others
 - can be changed by owner
 - "root" or "admin" or "superuser" has power to change permissions
- Email:
 - you can read your own email
 - can forward an email to someone else - this is also access!
- E-commerce login:
 - shopping cart etc visible only to user
 - financial information (credit card etc.) must be secure

Examples. Now, the Linux file system. So, hopefully all of you have got sufficient exposure to Linux at this point, that you know about file permissions. So, there are files for that matter even in a Windows or a Mac system, there is this concept of permissions, except that Windows and Mac are traditionally not multi-user systems so it is not very obvious upfront what these permissions refer to.

Now, on a Linux system, for example you have a file and the file has an owner associated with it. A file also has a certain group associated with it and this was something that was introduced early on in Unix based systems to provide the capability for multiple people to be able to access the file, but only specific groups. So, it is sort of a logical structuring but it has limitations.

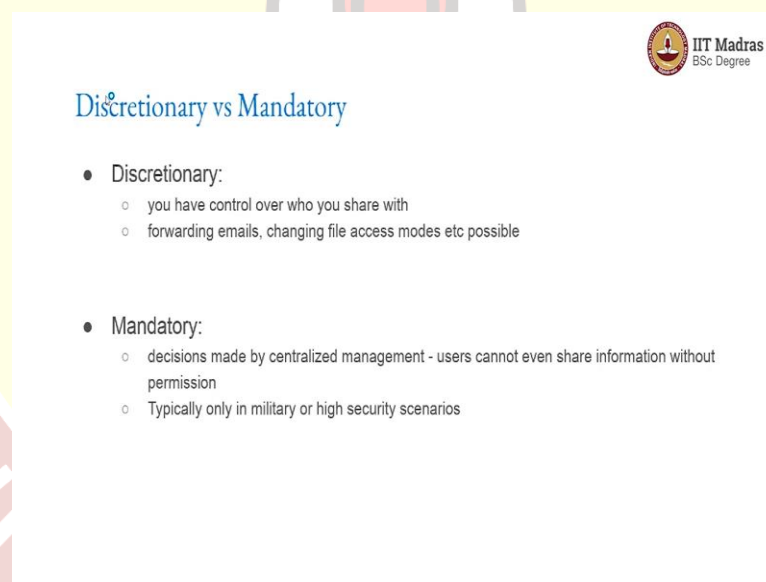
Now, typically, and the owner of a file can do everything with their files. They can access their own files, they can change the permissions, they can even grant access to others to be able to read or write or even delete a file if necessary. And there is also a so-called root or admin or super user who has permissions pretty much all over the system.


So, typically on a Linux machine the root user or admin user has complete permissions and they are, they could go in and read any file even if you try changing the permissions. And of course, like, the saying goes, with great power comes great responsibility, the expectation is that a person who is a super user or an admin is not going to misuse that power in order to go and create something of this type. But that is sort of you are trusting them not to do something bad.

Similarly, let us say, email. So, you have your Gmail account, you can read your own email. Now, the interesting thing is you actually have one extra permission which you may not even think of as a permission basically, which is that you can take an email that you receive and forward it to someone else. In some sense that itself is a permission, there can be systems where it sort of says, look you are not even allowed to forward this message to someone else. How can that be done, it is not easy to implement especially in the context of information. But that is a possible way of thinking about it.

And similarly, let us say you have an e-commerce website. You, the shopping cart should be visible only to the user concerned, and more information like the financial information, credit card, banks all of that must be definitely kept secure. You cannot have it in such a way that anybody can see anybody else's financial information. So, obviously there are different kinds of restrictions that you want to have on different types of data.

(Refer Slide Time: 06:21)



 IIT Madras
BSc Degree

Discretionary vs Mandatory

- Discretionary:
 - you have control over who you share with
 - forwarding emails, changing file access modes etc possible
- Mandatory:
 - decisions made by centralized management - users cannot even share information without permission
 - Typically only in military or high security scenarios

And, there are a couple of different ways of looking at this notion of access control. And one of them is what is called discretionary versus mandatory. And discretionary means it is left to the user's discretion, meaning that they have a choice. So, going back to the example of the Unix file system, or Linux file system, when I create a file I have a discretion or I have the power to change the visibility of the file.

Meaning that I can change the permissions so that anybody else on the system, even if they are not the super user can read the file. I can also change it so that they can edit the file if necessary.

And that usually also means they can delete the file if necessary. So, on the other hand since I am the owner of the file, I can also change it back so that it is no longer visible to someone else. Similarly, you have this power with respect to emails. You receive an email, you have the ability to forward it to someone else.

Now, there is another version of access control which is what is called mandatory. In mandatory access control, even these kind of permissions are difficult. If you have access to something that does not mean you can sort of pass that access on to someone else or change that mode of access in any way.

It is a very strict form of access control where everything, every change that needs to be made has to be in some sense authorized by a central authority, and as you can imagine this is most important in military kind of settings, where the secrecy levels and confidentiality levels are probably at their highest requirements. In most other cases, especially, with files and documents and so on, enforcing this kind of mandatory restrictions can lead to making life very difficult for anyone who is concerned with using it.

So, predominantly you are likely to come across discretionary forms of access control, and mandatory would be in some very restricted places. Maybe in the most productive trade secrets of companies or inventory, installations, maybe parts of banks or financial organizations and so on.

(Refer Slide Time: 08:43)



Role-based access control

- Access associated with "role" instead of "username"
- Example:
 - Head of department has access to student records
 - What happens when HoD changes?
- Single user can have multiple roles
 - HoD, Teacher, Cultural advisor, sports club member, ...
- Hierarchies, Groups
 - HoD > Teacher > Student
 - HoD vs sports club member? - no hierarchy here

Now, another sort of way of looking at this, this is, so all these are different ways by which we sort of understand the concepts in access control. Another set of concepts is what is called the role based access control. Now, going back to the Linux file system, over there, it is username based. So, I can basically, I am the owner because I am the, my username or user id is the one associated with ownership of the file. And if I want to grant someone else permissions on the file, I need to know their username, and I grant that to them.

On the other hand there is also a concept of something called a role. So, you might for example have a scenario where the head of the department, of a particular department in a college has access to student records, because they need to know who are the students who are having trouble, who need extra assistance with regard to assignments or courses or anything else, but, so what would happen is whoever is the head of the department would have access to all of the student records. But typically, in any college, the head of the department is for a fixed term and after some time it changes. Someone else becomes the head of the department. So, what happens when the HOD changes?

How should you have set the permissions on the student record? Should you have associated those permissions with the personal id of the person who was the head of the department, in which case you need to go and change all those permissions, take away the permissions from that person and grant permissions to the new person who is now the HOD.

Another better way of looking at it, in some ways is to say that a single person or a single user or a single entity can have multiple roles within the system. So, a single person could be a head of the department, but they are also a teacher, they might be in charge of maybe the cultural group and they are the cultural advisor or they may simply be a member of the sports club or gymkhana, which means that they have access to some of the gymnasium and badminton facilities, something of that sort.


And these different roles need not necessarily interfere with each other. So, do I have access to the squash courts on campus, that has nothing to do with my being faculty, or even if I am the head of the department of electrical engineering. What does matter is am I a member of the sports club. Now, you could have, with these roles, with this concept of roles in place you could have hierarchies. One example of a hierarchy is where typically an HOD has all the permissions

that a student has. So, you could say that an HOD is a superset, of a teacher, HOD has all the function of a teacher and therefore actually is a superset of teacher.

Similarly, a teacher could be considered a superset of student. Of course, I am sort of over simplifying here. You might want to have more fine grained permission control, but roughly you can see that this might be one logical way of looking at it. On the other hand, as I said in the earlier example, the fact that you are HOD of a department does not mean that you have membership in the sports club and therefore that does not automatically grant you access to the squash court or badminton court. So, you need to take that membership separately.

So, a role, by assigning a role HOD of a department, a teacher, a student, sports club member, each of those are different roles that can be assigned to an individual person. One person can be associated with multiple roles and obviously one role can potentially also have multiple people associated with it. There are restrictions. For example, only one person can be HOD of electrical engineering at a given point in time. But, multiple people could be studying in electrical engineering. So, there are different ways in which the role assignments have to be done. But if you now associate the access control with the role rather than the username, it becomes much easier to change and modify roles or to grant sort of fine grain permissions as required.

(Refer Slide Time: 13:03)



Attribute-based access control

- Attribute
 - time of day
 - some attribute of user (citizenship, age, ...)
- Can add extra capability over role-based

Another variant of this is so called Attribute-based access control. Now, attributes are some extra information which is associated with an entity, or not necessarily even with an entity, it could be with the system as a whole. An example of an attribute is the time of day. So, I might have

something which says that I am allowed to access certain machines only between 9 am and 5 pm on working days. So, that becomes attribute-based access control. I might have the role, I might have the permissions to access that machine, but there is some other attribute which says that the machine is kept inaccessible until it becomes 9 am on a working day, and after 5 pm it shuts off, I cannot access it anymore. So, you could add extra capabilities on top of role based access control by going with additional attributes.

(Refer Slide Time: 14:00)

Policies vs Permissions

- Permissions
 - Static rules usually based on simple checks (does user belong to group)?
- Policies
 - More complex conditions possible
 - Combine multiple policies
 - Example:
 - Bank employee can view ledger entries
 - Ledger access only after 8am on working days

And when you are implementing such access control mechanisms, there is also this concept of permissions and policies. Permissions are usually just static rules, they are based on simple checks. Thus, the file permission setting allow this username to read this file. Very easy to check. It is just one bit somewhere in the information which says okay, yes you have access or no you do not have access.

Policies can combine multiple conditions together. And an example was what I said earlier. You could have one policy which says that a bank employee can view ledger entries that are made in the accounting system, but there could also be another policy which says that ledger access is available to anyone only after 8 am on working days.

So, now in order to access this I need to be a bank employee and it needs to be after 8 am or an hour. So, I am combining multiple things together. All of that is expressed quite compactly and easily using a policy. If I just tried having fixed permissions then it is difficult to sort of see how do I create a permission which encapsulate all of these together. So, policies are can be sort of

layered together, and you can have multiple policies that apply to one particular piece of information.

(Refer Slide Time: 15:28)



Principle of least privilege

- Entity should have minimal access required to do the job
- Example: Linux file system
 - users can read system libraries but not write
 - some files like /etc/shadow not even readable
 - you can install Python to local files using "venv" but not to system path
- Benefits
 - better security - fewer people with access to sensitive files
 - better stability - user cannot accidentally delete important files
 - ease of deployment - can create template filesystems to copy

Now, in general one of the concepts that is applied in this whole area of access control is what is called the principle of least privilege. And put very simply, it says that any entity should have the minimal access required to do their job. In other words if what you need to do is basically just deliver an item to somebody, all that you need to do is know their address so that you can go, keep the item outside the door, ring the bell and leave. You do not need to, for example, go inside their house, find out what is inside the refrigerator, none of that. You do not have permissions to do any of those things.

Similarly, going back to the Linux file system again, users can read system libraries, because there are usually many sort of functions and so on that are compiled into library files, but you are not allowed to change those typically. And in fact, certain files like the /etc/shadow which stores the encrypted passwords, are typically made unreadable so that nobody, only the root, admin users can even read those files.

Now, you might have other kinds of permissions. For example, when you are installing python using the VM Virtual Environments, you can install it into your local file system, that is your own home directory, but you will not be allowed to modify the main system path, which is one of the reasons why the python VM was so popular. It allows you to create your own local thing without needing to mess around with the actual system.

Now, what are the benefits of this principle of least privilege? Fundamentally, better security. The fewer people have access to sensitive files, the better. How do you define sensitive files? Of course, depends on the context, but in general if I do not need access to something, I should not have access to it.

Better stability, and the reason behind that is if I unnecessarily provide access, especially write access to certain files, there is a chance that somebody even by accident like, due to a typing mistake might end up modifying a file or deleting a file. And that is it, your entire system stops functioning the way it was supposed to. It becomes unstable.

And another benefit is ease of deployment. You can basically create sort of template file systems, that can easily be copied from one place to another because you know that nobody else is allowed to modify it. Each person has only very restricted permissions, and it becomes easy to take it away.

Now, even though this is, principle of least privilege is easy to state, it is not always easy to implement in practice. And there are cases, for example, where you might find that, especially if you want to, if you are writing a web app and you want to accept file uploads from outside, you suddenly find that you need to make certain directories, sort of what is called world writable, meaning that anybody can write into those directories, and that actually messes up security in some way.

So, there can be things where just the nature of the framework or the software that you are using forces you to violate this principle. But as far as possible it should be followed. So, that is why it is not a hard and fast rule. It is more of a guideline or a principle that helps with, it has various benefits and is generally advised to follow it as far as possible.

(Refer Slide Time: 19:03)



Privilege escalation

- Change user or gain an attribute
 - "sudo" or "su"
- Usually combined with explicit logging, extra safety measures
- Recommended:
 - do not sudo unless absolutely necessary
 - never operate as root in a Linux/Unix environment unless absolutely necessary

Now, related to the principle of least privilege, is also this concept of privilege escalation. Meaning that normally I would perhaps work as a regular user, but there may be certain times at which I need to go and perhaps either add a new user or change some system file, or install a new software, and this is where, on the Linux system there is this concept which you might have come across called sudo.

So, sudo is used in order to perform one action with higher privileges than you normally have. There is also the su which allows you to completely switch user. You can actually impersonate another user. If you know the root password, then you can basically become root and from there you can also switch over to any other user in the system.

Obviously, a very blunt instrument. I mean it can be used to create a lot of damage if you are not careful. And it is usually recommended that you do not use sudo unless absolutely necessary, and do not use the raw su command at all. So, unless you have a lot of experience with these kind of systems, you should not be using it. And if you do have experience, most likely you will be so scared of what it can do that you will not do it anyway.

So, privilege escalation is something that has to be done with very limited cope. You need to do it as little as possible. And usually this is also combined with explicit logging. So, sudo, for example, logs every command that is run using sudo for the simple reason that if something goes wrong you should be able to go back and find out at which point it went wrong, and what was the mistake that happened.

So, all these are things to keep track of. I am explaining it in the concept of a file system, but of course you can sort of take these same ideas and translate them into the concept of a, into the context of an app, that you might be creating. Even there, let us say it is an e-commerce type of application, normally you are just a regular user of the system but sometimes you might have to switch over to an admin role to go back and look at if there are some entries at the back-end that do not make sense or somebody is saying that their shipment never came, what happened?

Should you always be logged in as an admin? That is dangerous because there is a chance you might accidentally go and change one of the entries, or you might click on a link which ends up modifying something which should not. So, logging in as an admin, or logging in as someone with higher escalated privileges is always sort of the last resort. You do it only when required.

(Refer Slide Time: 21:44)



Context: Web apps

- Admin dashboards, user access, etc.
- Gradebook example:
 - only admin should be able to add/delete/modify
 - users should have read permissions only on their own data

So, again, coming back to the context of web apps, you could have admin dashboards. Even in our Gradebook example, we are looking at a context where you might want to sort of add students, add courses, delete students or courses, modify the relationships between them and so on. And obviously in reality what we would want is that a given user is only a, given read permission to their own data. Admins are given permission to modify pretty much anything that they want. And you have to be careful about how you manage what you are doing. So, along with sort of enforcing logins and so on, it is also possible to add extra conditions over there which say, not only should you be logged in you also need specific permissions.

(Refer Slide Time: 22:35)



Enforcing

- Hardware level
 - Security key, hardware token for access, locked doors etc
- Operating system
 - filesystem access, memory segmentation
- Application level
 - DB server can restrict access to specific database
- Web application
 - Controllers enforce restrictions
 - Decorators in Python used in frameworks like Flask

So, this brings us to one question, how do we enforce such permissions or such restrictions? There are multiple layers at which it can be done. One of them is at the hardware level, the most fundamental, so the hardware level. And the simplest way to think about it is if I want to enter your room, I need to have a key to the room.

So, that is a physical piece of metal that I need to carry with me which grants me access to the room. In the same way, in the context of information security, there are usually some kind of hardware tokens, USB tokens or NFC cards, the various kinds of things that are created these days, which the person needs to have that in order to be verified at some point so that they can grant access.

But typically, more than the hardware level, the place where most of the permissions and so on are enforced in the context of information technology are at the operating system. So, there will be file system level permissions, there will be memory segmentation so that one process cannot read something from another process. The operating system essentially ensures that, whole principle of least privilege. Each process or each user is given access only to what they need. And if you need access to something else you need to work with the creator of the other application to find out how to share information between the two.

Now, as a developer, as a web app developer you would probably come across application level control, where for example, you might find that you are sharing a database server with other

people, and the database server might then restrict access, restrict your access to one specific database within the system.

And as a web application developer you might find that you want to be able to restrict which parts of the app different users get access to. And if you think about it, given that we have been working with the NPC model all along, the controller is the ideal place to enforce a restriction because after all, what we have is, we want to be able to restrict access to certain views, but how are views generated? They are ultimately some kind of routes in the application which take a certain url and calls a certain controller function, and that function then needs to basically pull up the appropriate view.

So, in order to, when you are executing that function, if I can somehow put a wrapper around it which says that only if some condition is satisfied should I continue with the rest of the execution of this function, then that should take care of my program for me. Now, how do I put a wrapper around a function and enhance its functionality? We have already seen examples of that. For example, the `@app.route()` kind of notation that we use in flask is a decorator, and decorators in python are an ideal way of sort of enhancing the functionality of a given controller, or a function. We will see examples of that later.

