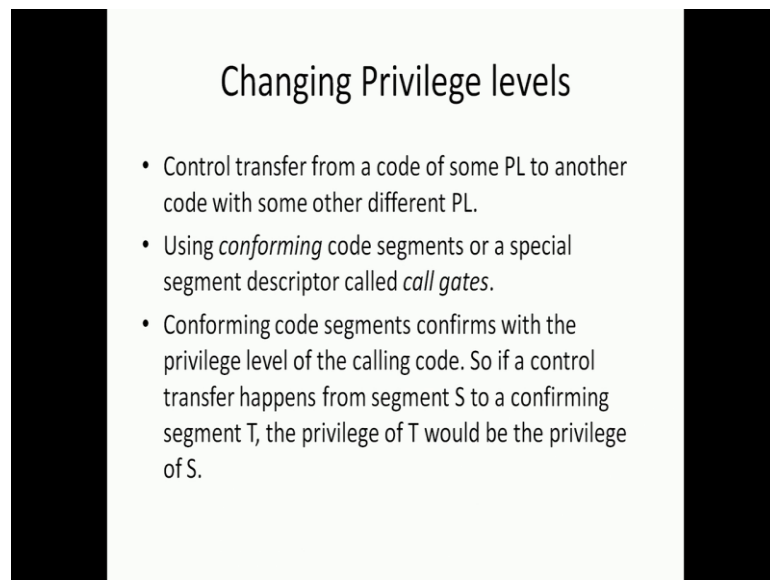


**Information Security - II**  
**Prof. V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 19**  
**Architectural Aid to Secure Systems Engineering**  
**Session - 18: Memory Segmentation Deep Dive - 4 (Changing Privilege Levels)**

Now, when we saw the types of segments in some of our earlier sessions, there were something called a conforming segment, conforming code segment and then, we have been talking again and again about call gates. Those two parts are very important from a security point of view because those are the things that are going to allow a numerically higher level code segment to transfer control to a numerically lower level code segment. So, these are the two important constructs that can make or kill security. So, let us understand this in a proper perspective and when we actually do an analysis of an operating system in terms of security, specifically from an x86 point of view, these are the things that come very deep into it.

(Refer Slide Time: 01:13)



**Changing Privilege levels**

- Control transfer from a code of some PL to another code with some other different PL.
- Using *conforming* code segments or a special segment descriptor called *call gates*.
- Conforming code segments confirms with the privilege level of the calling code. So if a control transfer happens from segment S to a conforming segment T, the privilege of T would be the privilege of S.

Let us take this. There is a control transfer from a code of some PL, some privilege level, to another code with some other different PL. We have talked about same PL already. Now, we are talking about one PL and another different PL. Now, we can use conforming

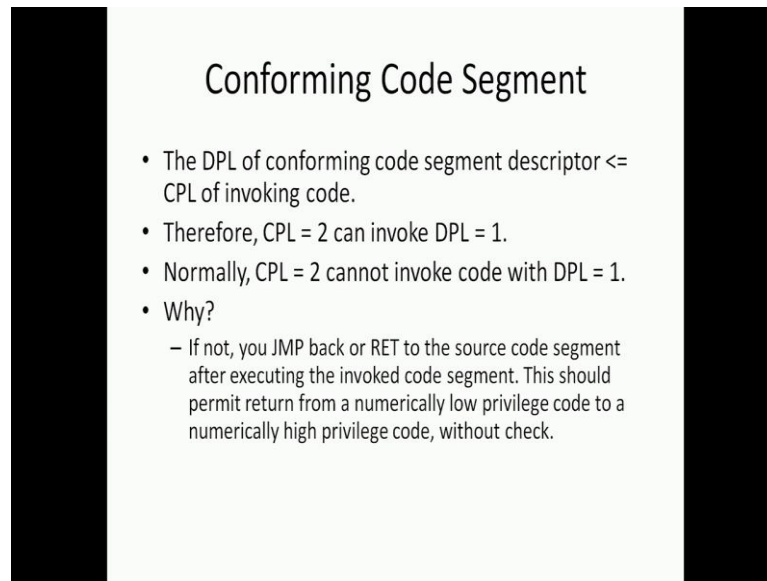
code segments for doing this or a special segment descriptor called call gates. We will see, what are those things as we proceed in this session.

First thing is, what is this conforming code segment? See, the conforming code segment is a code segment but it will have its own privilege level, which is do not care. The adjective conforming itself says it conforms to something else. What does it conform to? What do you mean by conforming to something else? Let us look at in detail.

So, if a control transfer happens from segment S, some segment S to a conforming segment T, the privilege level of T would be the privilege level of S. So, the conforming code segment confirms with the privilege level of the calling code. So, if a control transfer happens from privilege segment, from segment S to a conforming segment T, now the privilege of T will be the privilege of S. You are getting this point. So, now t will start executing as if it is a privilege level three code. If you are privilege level three and you call me I am a conforming code, I will start dancing to your tune. You are privilege level 2, you call me I will start dancing to your tune. So, now this is very interesting.

Now, again, I leave it as a simple exercise for you to find out where we will be using these things, from where will conforming code segment becomes become important. I will just leave it as a simple exercise. Think about it. So, these are the ways, you know, you may come out with a different answer than what I have.

(Refer Slide Time: 03:36)



### Conforming Code Segment

- The DPL of conforming code segment descriptor  $\leq$  CPL of invoking code.
- Therefore, CPL = 2 can invoke DPL = 1.
- Normally, CPL = 2 cannot invoke code with DPL = 1.
- Why?
  - If not, you JMP back or RET to the source code segment after executing the invoked code segment. This should permit return from a numerically low privilege code to a numerically high privilege code, without check.

Now, the DPL of conforming code segment descriptor should be less than or equal to the CPL of invoking code. This is very interesting. Therefore, CPL equal to 2 can invoke a conforming code segment with DPL equal to 1. Normally, CPL equal to 2 cannot invoke code with DPL equal to 1. But, if it is conforming I can invoke it with DPL equal to 1. But, CPL equal to 2 cannot invoke a code with DPL equal to 3 because that means, the three, I have myself classified it as a three level code. And, now it starts executing at privilege level two. I do not know what will happen.

So, please note that even if I am talking about conforming code, I am putting lot more restrictions there because the conforming code is written by me as an operating system. So, I know what it can do. So even if worst case, the three fellow comes in, I have privilege level two and I have done lot more of stuff.. So, I am having. So, a conforming code segment has strength of being in privilege level two.

So, if it is privilege level three, I do not care. So, a three fellow can come and access me. Please note, this is a very conceptually different part. But, when I am waiting a conforming code segment, which is at privilege level three, it has the strength of being only at privilege level three. It cannot execute at privilege level two. That is what I am

stating as an operating system. When I say the descriptor for this code segment is 3, essentially means, this conforming code segment can work only at 3.

Now, when a two level fellow calls that fellow will starting executing a privilege level two which I do not want. So, the descriptor privilege level of a conforming code segment should be less than or equal to the current privilege level of the invoking code. This is because I have written the conforming code segment. And, I have given it enough strength to be at that level, so anybody above that level, a numerically higher than that level comes and executes it, still it can stand that strength.

But, anybody from lower cannot be executed because this segment itself, according to my definition, is not capable of executing at a level lower than this. And, that is where your DPL plays a very close role. This is another big role for the descriptor privilege level, where the operating system can express itself saying what is the strength of this code, in terms of which privilege level it can execute? Are you able to follow this?

So, that whatever I have said, Why CPL equal to 2 cannot invoke DPL equal to 3? That is what. If not, you jump back or return to the source code segment after executing the invoke code segment. This should permit return from a numerically low privilege code to a numerically high privilege code, without check.

So, essentially this is; if you read this again and again, you will understand what I have. If you have followed what I have lectured, then neglect this sentence. If you did not follow, you know, look at the video plus this. So, you will follow that. But, do you all understand what is conforming code segment? Now, comes the next. So, what is this trying to do? This is one way by which I could come from a numerically higher privilege to a numerically lower privilege.

(Refer Slide Time: 07:21)

### CALL GATE descriptor

- Is defined by a system descriptor (S=0) in GDT which is used by the JMP or CALL.

The 64-bit descriptor in GDT

Destination offset 31-16	P, DPL	0110 0	000	WC	Destination Selector (16 bits)	Destination Offset 15-0
-----------------------------	--------	-----------	-----	----	--------------------------------	----------------------------

- Not only the selector for the target code segment, but also the offset in the code segment from which you should start executing is specified. The source code segment can only use it like a black-box

There is another way of coming from a higher privilege to a lower privilege, where in this is not conforming. So, the lower privilege fellow will work at that privilege level only. In the previous case, what I did? The lower privilege level, the conforming code segment will work at the privilege level of the higher fellow.

So, if I am a PL 2 code, you are a PL 3 code calling me and I am conforming. I will start working like PL 3. But, in this call gate I will work only as (Refer Time: 07:51). You can call me and I will start working at PL 2 and I will give you the answer. But, I will not change my privilege level to three and work. So, this is the difference between conforming code segment and call gate.

Now, what happens in this call gate? This is the descriptor of this call gate, which essentially gives me a selector, which is sixteen bits. Please, note the second field. This is the selector of the code segment and then, the offset within that code segment, which is thirty two bits. And, there is a P, DPL here. This is the system descriptor and there is a P, DPL; which is the present bit and the DPL bits.

Now, what happens is that DPL can tell me, this is a descriptor available in the GDT. It cannot be in the LDT. So, please note. This cannot be in a local descriptor table. This will

be in a global descriptor table. Again, there is a level of protection there. Now, what is this say? You are a PL 3 code; you want to execute some PL 1 functionality. May be a system call, you want to execute. Then, what you do? I as an operating system has designed that system call. So, when I design that system call, I have taken lot of precautions. So, that information does not leak or you do not do any damage when you execute me.

Then, I give you an interface. That is called call gate. In that call gate, so, I am a code segment, I have a descriptor. I will store that descriptor in that call gate. That is called the destination selector; the index of that descriptor in that destination selector. Correct. And then, I will store an offset within that where you can start executing. So, when you just call this particular index in the GDT or you jump to this particular index in this GDT, what will happen? This particular destination selector will be taken, that code segment will be taken that code segment base will be taken. To that base, this offset will be added and you will start executing that off set. And after you finish, you go back to the original thing.

So, what I have done here? I have basically said that I am a routine, I have very high privileges and you can use me. But, you have to come through that gate. You cannot use me as it is. You cannot use everything of me. You can come through a specified call gate. And, in that call gate it will give access to me but only to some part of the code because I have put the offset there. So, you can start executing exactly from that off set and go back.

So, I also define an entry point into my code. You cannot come and arbitrarily enter my code. I will define an entry point. From that entry point, you can (Refer Time: 11:08) and go off. And, when that is getting executed from that entry point to that, it will work in my privilege and you will go back.

So, can you give me a very interesting example for this. An interesting example for it is password. Now, password is to access the file, which is read and writable only by the administrator. You cannot read and write know. If you look at the password, you know, if you look at these access permissions, access rights for a password file, a normal user

cannot write and read. But, what you are doing? You are changing your password. So, you are actually writing into that file. Correct. But, only the user, only the super user can have read and write permission for that file.

How are you able to go and access and read and write into that file? You are reading your password also. How? When are you reading your password, you would try something. It will get hashed, and that hash is also get from there. Similarly, you are writing into your password. How is it possible though you do not have read write permission? This is a way I am just telling. May be, I do not know how Linux has implemented this. But, this is the way we can implement. So, there is a password executable program, which is user level program. Now, that is stored in some code segment. Now when I type password, that particular program there, it offers a call gate. So, I will go to that call gate and there is some specific things I can do. I will go and that will start executing that password program from that particular entry point. And that call gate, it is defined by the operating system. It will start executing from that point. It will execute. It will change. It can change because it is now privilege level zero. It can change the file and it will give me back.

So, it will take value from me, through the call gate it will go there. Go and execute that program, do whatever things at that privilege level and come back. So, this is very very important. Otherwise, every time I have to change the password. I have to tell the password to the root and he will change it for me. And your root is friend, code and uncode you are done for. So, this is very very important. So, this call gate descriptor; this is one example where we can use call gate. I do not know whether operating system uses this or not, but this is very interesting there.

So, what we have done in this set of three to four slides is that how will your privilege level  $k$  code call a privilege level  $k$  minus one,  $k$  minus 2 code. And, please note that we have lot and lot of privilege checks that happen here. It is not that you just allow it.

And, when you really want this whole thing to be much secure all the conforming code segments, all the segments that are subjecting itself to a call gate type of access should not be changed. So, you sign these functions also. You sign these functions, so that they

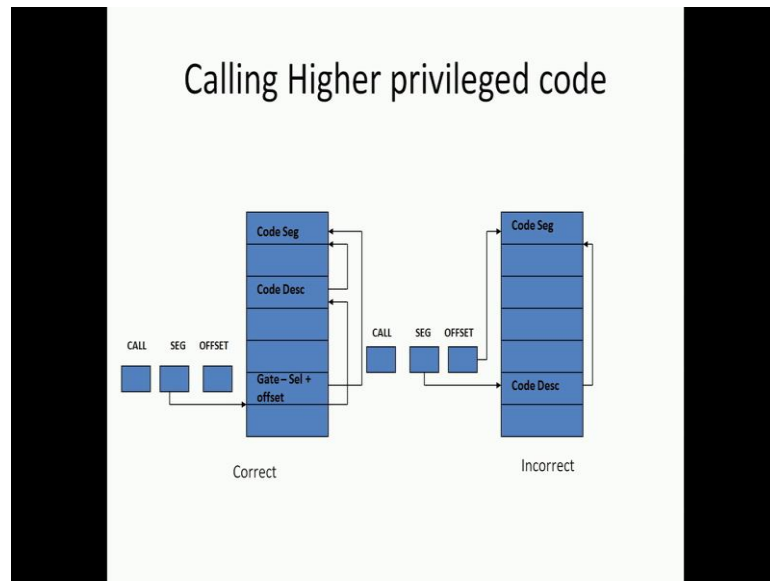
never changed. You have done it. You have done some verifications saying that it is secured. Now you sign it, so that every time it gets executed, the sign is basically being checked. So, there cannot be some modification. This is of course, which will allow you some penetration here. But, these are all very important functions especially, the call gate.

Now, another interesting point about both this call gate is there is a DPL there, if you look at that. So, this call gate cannot say just I have a call gate, anybody, any Tom, Dick and Harry cannot come and access it. If I make DPL equal to 2, right, only 2, 1 and zero can use my call gate. If I make DPL equal to 1, 3 and 2 cannot use this call gate. One can use it. And, the other code will be zero. So, one can use to come and execute to zero. If DPL is 2, 2 can come and they execute something in 1 or 0. If it is 3, then anybody can come.

So, I also have one more level of protection there using that DPL, which says that though I am a call gate and I am open for higher privilege levels to come and execute, to come and ask me to execute from some point, from that given point. But, it is not that every fellow can come. So, I have some control there through this DPL. So, so many small interesting things are part of this, you know, privilege switching which we need to understand and implement it.

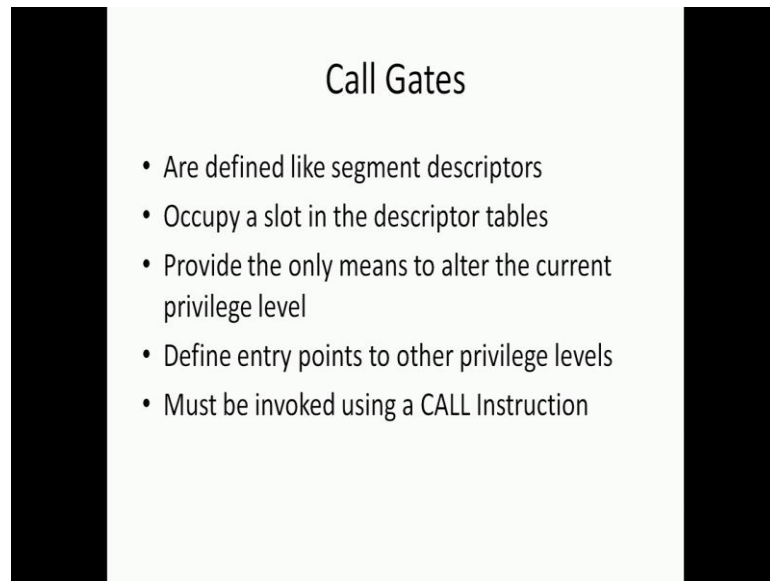


(Refer Slide Time: 16:00)



So, calling higher privilege level code. This is what? I have just summarized what is this. So, I directly will not go to the code segment and code segment descriptor. And then, I am talking about the incorrect path. The segment is not directly to the code descriptor. It does not point to the code descriptor, rather it points to a gate. And, that gate will tell you what is selector plus offset? And that from the selector, I will get actual code segment a little. So, there is a gate which will tell me which code segment I should use and what is the offset. So, there is one protection that is given here. I am not directly going to the code descriptor and going to the segment and start accessing. This is not possible for going, calling higher privilege means, numerically lower privilege code. So, this is, this describes the call gate.

(Refer Slide Time: 17:08)

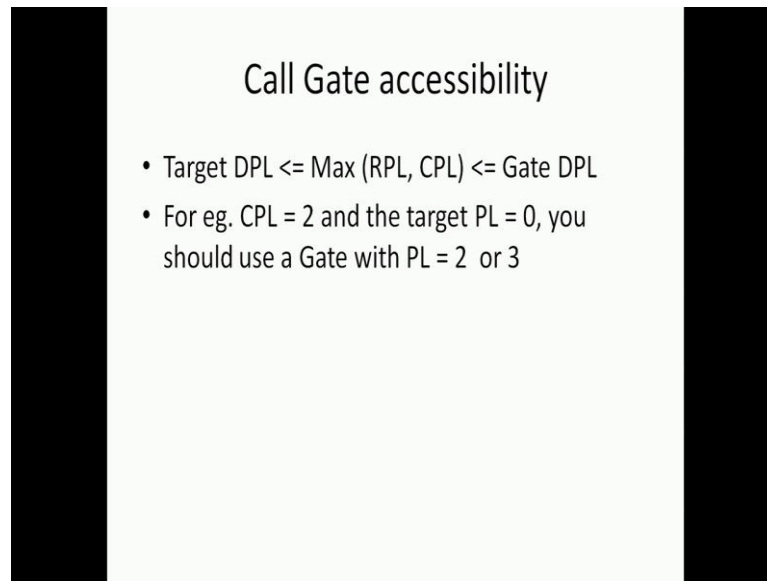


## Call Gates

- Are defined like segment descriptors
- Occupy a slot in the descriptor tables
- Provide the only means to alter the current privilege level
- Define entry points to other privilege levels
- Must be invoked using a CALL Instruction

So, the call gates are defined like segment descriptors. They occupy a slot in that descriptor tables. They provide the only means to alter the current privilege level. And, they also define entry points to every privilege level. So, if you are privilege two, I can go here privilege one. You can; so, I can put different call gates for different things. So that, you know I can have different entry points here and must be invoked using a call instruction because I have to return back. He cannot jump to me. I know, I should know how to send you by.

(Refer Slide Time: 17:46)



### Call Gate accessibility

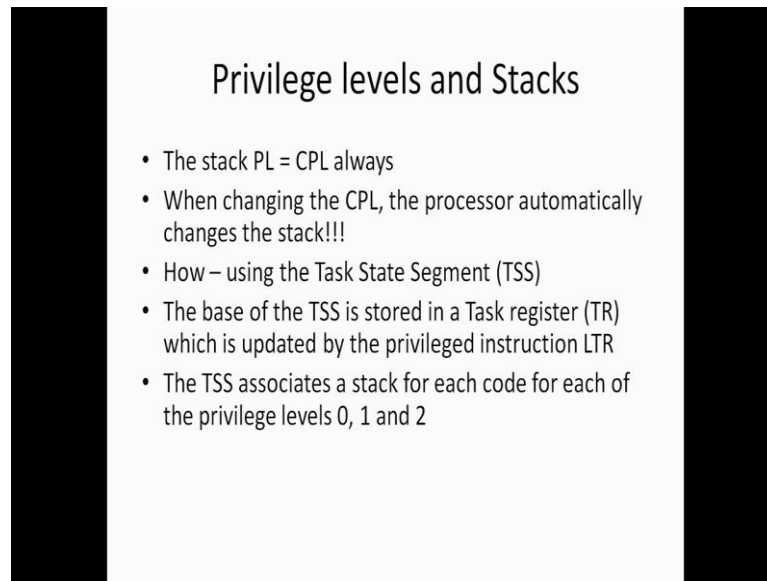
- Target DPL  $\leq$  Max (RPL, CPL)  $\leq$  Gate DPL
- For eg. CPL = 2 and the target PL = 0, you should use a Gate with PL = 2 or 3

So, please note here. Your call gate accessibility is your target DPL. Target; so, there are; please, note very carefully. There is a gate, which has a DPL. There is a code descriptor, which has a DPL. First, you come to the call gate. There is a DPL there. Then, there is a code descriptor that has a DPL, and there is a CPL, this is called the gate descriptor privilege level. Whatever is there in this code that is called the target descriptor privileged level.

So, there is a current privilege level, which I am executing. There is a gate DPL. There is a target DPL of that. So, your target DPL should be less than or equal to max RPL, CPL, which should be less than or equal to a gate DPL. So, I can only through a call gate go to a descriptor privilege level. That is less than me or max of RPL, CPL. And to go to that, I go through a gate. And that gate should be at a descriptor privilege level, which is larger than me.

So, for example, if CPL is equal to 2 and the target PL equal to 0, you should use a gate with PL equal to 2 or 3. Then only it will allow you. If you use PL equal to 1, the gate will not allow you. So, gate will check you if do you have enough privilege and the target should be at a lesser privilege than or lesser or equal to privilege level, you cannot jump again. For the same reason, I cannot use a call gate to go to a higher privilege.

(Refer Slide Time: 19:45)



### Privilege levels and Stacks

- The stack PL = CPL always
- When changing the CPL, the processor automatically changes the stack!!!
- How – using the Task State Segment (TSS)
- The base of the TSS is stored in a Task register (TR) which is updated by the privileged instruction LTR
- The TSS associates a stack for each code for each of the privilege levels 0, 1 and 2

The stacks privilege level will always be equal to the current privilege level. As I told you, when I like to load a stack segment, the privilege level should be equal to exactly the same privilege level. So, when changing the CPL, the processor automatically changes the stack. Why it has changes the stack? Where will it get the stack? For every privilege level, there is a stack available in the task state segment. And, we have already seen why this is important because from a security point of view, this become extremely crucial.

So, the base of that TSS is stored in a task register. Task register points to, which is updated by the privileged instruction what you call as LTR. So, TR actually points to; what will it point to? It will point to TSS selector in the table. From there, it will point to a TSS descriptor, which will point to the base of this. So, essentially from the task register you can basically get the base. The task state segment associates the stack for each code for each of the privilege level 0, 1 and 2. So, that is very important.