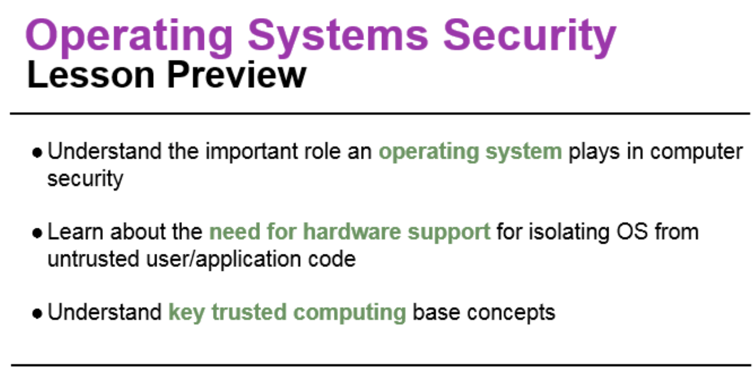
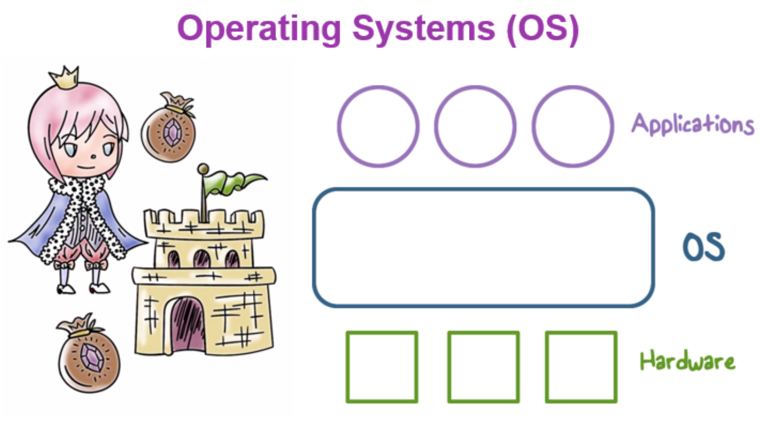


Operating Systems Security
Lesson Preview

- Understand the important role an operating system plays in computer security
- Learn about the need for hardware support for isolating OS from untrusted user/application code
- Understand key trusted computing base concepts

The operating system plays a really critical role in protecting resources in a computer system. Resources such as memory pages, or files that might contain sensitive data. It makes use of hardware functionality and uses that to isolate itself from un-trusted user code, and also can isolate one user's process from a different user's process. So we're going to see how that is done by the operating system.

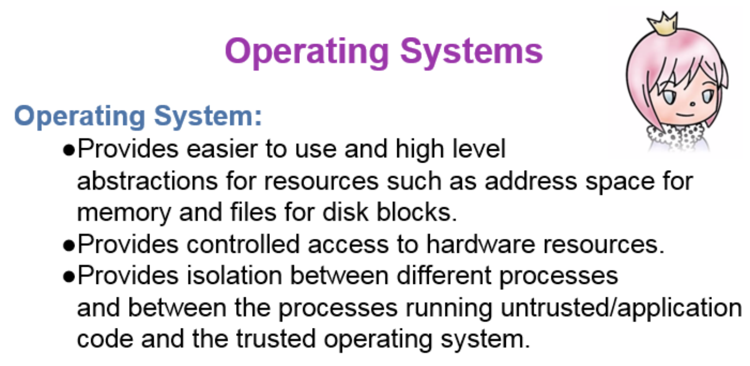


Operating systems play a really important role in computer systems. In fact, when we talk about computers, we often say the operating system running on a certain kind of computer, a Windows machine, or an iOS device. We're going to see that operating systems play an extremely important and critical role when it comes to protecting and securing resources that we have in our computer systems.

So let's look at what a computer system is actually. So we actually start with the hardware. Your hardware consists of a CPU, perhaps you have memory, and other IO devices, so that's the hardware that we have.

Now if you wanted to use the hardware directly, it's going to be pretty difficult. In fact no one does that. What we really do is run a really important program, actually operating system is kind of a program that handles the low level hardware resources that we have. So your operating system may be Windows or Android, or a number of these operating systems. What we generally start with the hardware run the operating system on that device that we have. And the applications that we directly deal with, that we run on our systems, so this could be your browser, let's say, or this could be your mail client. So these applications then actually running on top of the operating system.

One way to think about this picture that we have for a computer system is that hardware is where we have the real or physical resources. We're going to run an operating system to control access to those hardware resources, and those resources then are going to be made available to the various applications that we have. So, if any of these resources had to be protected in case we have different applications running here. The reason the operating system plays a really important role is because the operating system gets between these applications and how they can access the resources.



Operating Systems

Operating System:

- Provides easier to use and high level abstractions for resources such as address space for memory and files for disk blocks.
- Provides controlled access to hardware resources.
- Provides isolation between different processes and between the processes running untrusted/application code and the trusted operating system.

Applications rely on the operating system. So what does it really do for you?

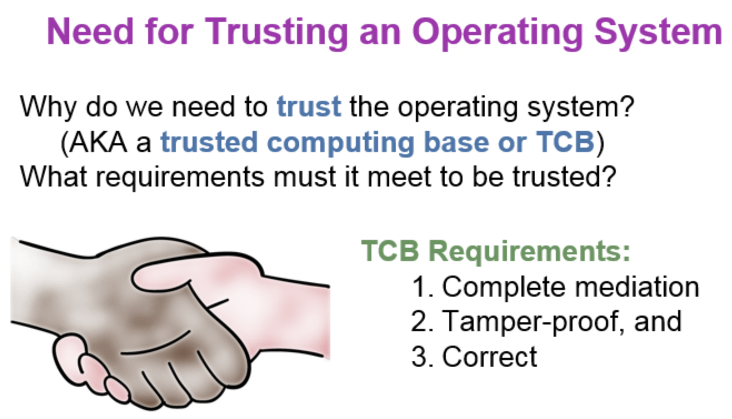
First of all like I said before, the hardware is actually not very easy to use if you had to use it directly. If you wanted to program your applications, and had to work with the hardware directly it's going to be infinitely more complex. It may be already be hard, but it's going to be much harder. So what the operating system does is that it creates easier to use and high level abstractions for the resources that we have that the hardware provides.

For example, we know that data that must be persistent, it's stored on the disk in disk blocks. Well we actually don't directly work with disk blocks and keep track of where on disk a certain piece of information may be. We have things called files. Think of file as a high level, where we're going to talk about virtual resources that are created or supported by the operating system and made available to you. So these high level abstractions that we can use to access more user-friendly resources in some way, make it easier for us to write our applications, or build our applications that are going to run on this computer system.

Now the resources that we're going to provide, the high level resources, the operating system makes available to the applications. Well, they going to be implemented using the real physical resources that we have that the hardware provides to us. The resources, the physical resources are going to be shared across these different applications. Obviously when you have that kind of sharing, we need to access those resources in a controlled fashion. So the hardware resources are actually managed by the operating system, and access to those is going to be controlled via it as well. And that actually is one of the fundamental reasons why operating systems have such an important role to play, when it comes to protecting resources and securing access to them.

The last thing that is really interesting and we're going to spend a good bit of time on that, is all the different applications or different processes going to run on the same system and share the resources we have. The physical resources that we have between them, but the operating system makes each process believe as if it's the only one running. And it's able to do that by providing what we call isolation between these different processes. In some sense one process or one application does not need to be aware of another application, unless it explicitly decides to interact with it. It's sharing these physical resources, but think about the operating system making sure that what is being used by one application doesn't get used by somebody else, or another process or another application.

So, having this isolation, sort of giving each application, sort of the feeling that it is the only one running. We should say that if they share, of course, not each process has all the resources, so there may be some performance implications. But, if you don't worry about performance, basically each process can believe as if it has the resources, and the computer to itself. So, the operating system is actually going to create this isolation between these different applications, which is really important. Because applications may not trust each other, and if they don't trust each other, they don't want any sort of interference from other processes that are not trusted by them. So this isolation that the operating system provided isolation is actually going to guarantee to us that the process doesn't have to worry about other processes, or other applications that may be there in the system.



Need for Trusting an Operating System

Why do we need to **trust** the operating system?
(AKA a **trusted computing base or TCB**)
What requirements must it meet to be trusted?

TCB Requirements:
1. Complete mediation
2. Tamper-proof, and
3. Correct

Why do we need to trust the operating system? And I'm going to talk about exactly what does it mean to trust a system, but that sort of differentiates an operating system from other applications that may not be trusted. So, why do we need to trust it? [note: Udacity video reverses #1 & #2 from .pptx – order here matches Udacity]

In fact, not only we need to trust it but we actually also call it a Trusted Computing Base. So let's see why we call it a TCB or a Trusted Computing Base.

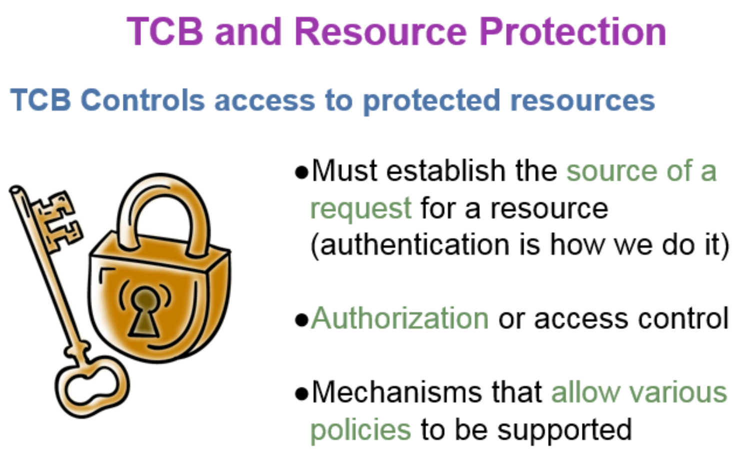
It is the base in some sense. If you don't directly want to deal with the hardware, we set applications on top of the operating system. So the operating system really is the computing base that is seen by the applications or the user processes. The need to trust to the operating system comes from the fact that we giving it the keys to the kingdom. Keys to the kingdom here are direct control of all the physical resources. So the operating system in some sense is able to access anything that we have and then it's the job of the operating system to make sure that these resources, or whatever, high level resources we implement using the low level physical resources get accessed by the correct users in the system or the correct applications. And if you're going to make someone sort of the in charge or the controller, you better be able to trust that it's going to do what it's supposed to do. Otherwise, the results that you going to have, you will not like them. So that's why there's need to trust an operating system.

So what the exactly does it mean for us to be able to trust the operating system. So what are the requirements that should be met by this Trusted Computing Base that we say sets an important role in protecting our resources?

1. First of all, the operating system absolutely has to be between the untrusted applications and the physical hardware resources that we have. This is called complete mediation. Think about the request coming from an untrusted application and this request is for a resource that is being requested by the application. It has to go through the operating system. The operating system has to mediate this request and you can think about why that's necessary. It's necessary because we had the operating system has to check that the resource of the request actually has access is allowed to gain access to this resource. And for that check to happen, the operating system has to become between the request and the resource. So that's called complete mediation. Okay, you can't get to any resource that need to be protected without actually going through the operating system. So trusted system is always going to take a look at any request before it actually reaches the resource, and the source of that request is able to access that resource. So the complete mediation requirement is the first one.
2. The second requirement a Trusted Computing Base has to meet is, what we call, it has to be tamper-proof. In fact, we going to spend bunch of time how to achieve that, or how to meet that requirement. The reason it has to be tamper proof is that we're talking about untrusted

code and in the Trusted Computing Base of the trusted operating system. Well, if the untrusted code can tamper with the trusted operating system, you can't trust anymore. The untrusted code tampers with it, changes it to do whatever it wants done. In particular again, access to resources that perhaps were not meant for this untrusted application, which may be compromised, could be malicious.

3. Third requirement is the so called correctness requirement. If you going to call a system trusted, you going to completely rely on it to make sure that you're protected resources get used in a proper way. Then however it does that, whatever functionalities implemented by the Trusted Computing Base, it should be done correctly. Because if there is a vulnerability or a bug or some error in the Trusted Computing Base or the operating system. I said it has the keys to the kingdom. So, we said operating system has to be trusted. Now we know what it means for it to be trusted. You shouldn't be able to bypass it. Okay? That's complete mediation. You have to go through it. You can't alter or change it, if you are an untrusted application. And whatever functionality it implements, that's done correctly.



TCB and Resource Protection

TCB Controls access to protected resources

- Must establish the source of a request for a resource (authentication is how we do it)
- Authorization or access control
- Mechanisms that allow various policies to be supported

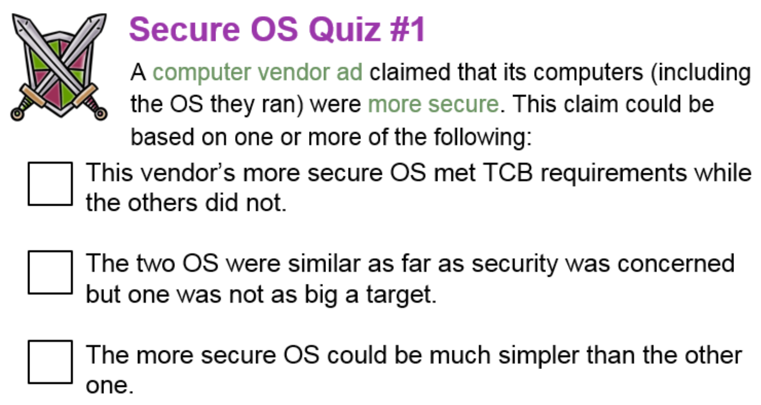
One of the things the operating system does is protect resources that need to be protected. How does that work? [note: Udacity video lists "OS" rather than "TCB" in .pptx]

Well, the way it works is because we give control of the resources to the operating system. We say you can't directly, untrusted applications can't directly access it. So, operating system has control and it's actually going to control access to these resources, and we know it can do that because of the complete mediation requirement that we have. So, anybody who wants access to such a resource actually has to come through the operating system and when they come through the operating system, the operating system is going to be able to say yeah this is okay or I'm not going to allow or grant access or I'm going to deny this particular request. So how does the operating system actually sort of mediate these requests that we're going to have for various resources and that needs to be protected?

First it must establish who's making the request. What is the source of the request? Request always has a source. The entity that's making the request and it has a target which is the resource that is being requested. So the operating system has to know who is making the request and what is being requested. So the who part is typically answered by this thing we called authentication. We're going to talk more about it in a future lesson. But authentication essentially establishes on whose behalf a certain application or a process is running. So that's the user who's making the request.

Once we know who is making the request then we have this thing called authorization or access control. This is really checking, looking up if this source of the request is allowed to access the resource for which the request is being made, okay. You may be allowed to access a certain file or you may not be able to access a file because it doesn't belong to you. So the check that we're going to do is called the access control check.

So the trusted computer base of the operating system really does It doesn't decide who accesses a given resource or not. That's called select policy. You have to decide who you're going to share your resources with or what other applications can share a given file that you have. So that's the policy. But the fact that when a request for the resource comes, being able to do that access check. The mechanisms are essentially that part which intervene between the source of the request and the check that we going to do to determine if the policy that we have in place is going to tell us whether the resource can be granted or not. So the mechanism is essentially this check that we are performing. And the goal of Trusted Computing Base is to implement and do that in a structured fashion the checks or how the mechanisms for doing these checks that going to allow a variety of policies to be implemented or supported in the system. The protection of the resources when you talk about it always we have to know where the request is coming from. Whether the requester is allowed to access it and that's what the Trusted Computing Base has to do and it does it by providing these mechanisms to support variety of access control policies.



Secure OS Quiz #1

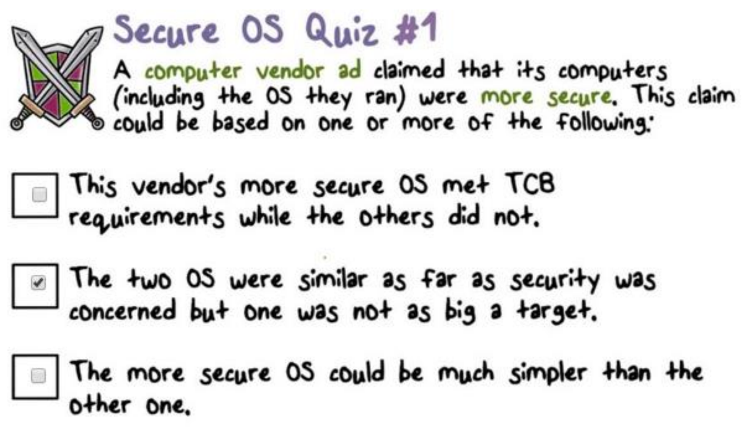
A computer vendor ad claimed that its computers (including the OS they ran) were more secure. This claim could be based on one or more of the following:

- ☐ This vendor's more secure OS met TCB requirements while the others did not.
- ☐ The two OS were similar as far as security was concerned but one was not as big a target.
- ☐ The more secure OS could be much simpler than the other one.

Let's talk about a question that actually comes from an ad that was run on TV by a well-known computer vendor, I will not name it. But that ad basically claimed that its systems didn't suffer from any of the ailments that the competition system had to worry about. So this claim that was made by this vendor who said it's system was superior, and when it came to security could have been well founded, or it may have been a marketing thing. Here we want to sort of dig deeper and to sort of think about vendor A and vendor B, and they both supply operating systems. And vendor A's making this claim about better security that its system offers in some sense. So what could be the basis of that claim?

So we haven't talked that a lot about how one would validate this sort of a claim, but we know what a trusted system is and what requirements it must meet. And we also talked about the threat model that is who targets these kind of systems, and one of our design principles said it is easier to get things right when they're simpler, okay. Complexity is the enemy, leads to vulnerabilities that can be exploited and so on.

So I want you to use your knowledge about threat modeling, about design of secure systems, and the trusted computing system requirements to think about the options that we have here and check the ones that you think there is some reason to believe that it may be true. Once you do that, we'll come back and talk about the answers.



Secure OS Quiz #1

A computer vendor ad claimed that its computers (including the OS they ran) were more secure. This claim could be based on one or more of the following:

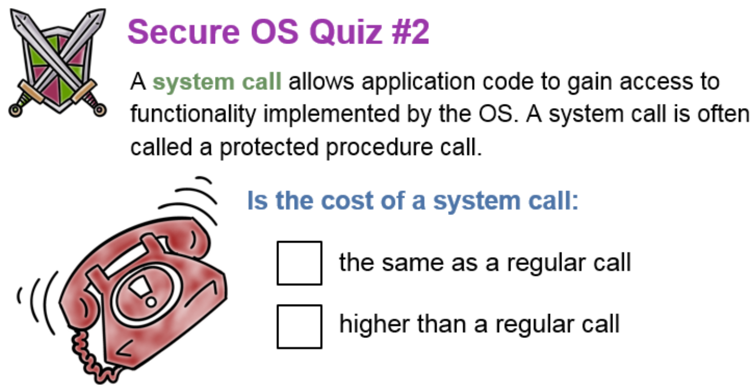
- [ ] This vendor's more secure OS met TCB requirements while the others did not.
- [✓] The two OS were similar as far as security was concerned but one was not as big a target.
- [ ] The more secure OS could be much simpler than the other one.

Okay, let's talk about the options that we have here. So it is true at the time this ad ran, this is a few years ago, that vendor A, its systems didn't have as many, sort of, reported attacks as vendor B's. Or maybe you can call this vendor A and vendor m and you can figure out who B is and who M is.

So it's true that it didn't, but both of the systems were running complex rich operating systems that supported similar kind of applications. So both tried their best to meet the requirements that we listed for a Trusted Computing Base. So it's not quite true that one vendor's operating system was significantly more secure than the other vendor.

Okay, let's look at the second one. The second option says, the two operating systems were similar as far as security was concerned but one was not as big a target. So this is about threat modeling that we're talking about. Who is that coming off to you? And it is too at that time vendor A's systems were not as big a target as vendor B's, because vendor B actually had most of the market belonged to it. So the attackers obviously were going after these kind of systems, and vendor B's system was a much bigger target than vendor A's. One reason was, as we said, market shares. So in this case, I'm going to say, this is one of the reasons why the claim that vendor A's systems didn't suffer as much as vendor B's was true, because vendor a's system was not as big a target, okay?

So the last option says, the more secure OS could be much simpler than the other one, okay? So this is our design principle for secure systems. So that's what I want you to think about. But I said both systems are actually full faced operating systems. They supported same kind of applications and neither one of them actually we can say was much simpler. By the way this used to be an ad that Apple ran. And the sickly system that suffered from all the ailments was a Windows system. So I gave it away.



Secure OS Quiz #2

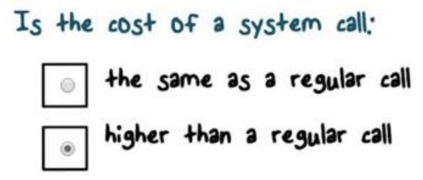
A system call allows application code to gain access to functionality implemented by the OS. A system call is often called a protected procedure call.

Is the cost of a system call:

- [ ] the same as a regular call
- [ ] higher than a regular call

Okay, so actually I haven't told you, so far, we're going to talk about it, what a system call is. But let's sort of think about what would be a system call.

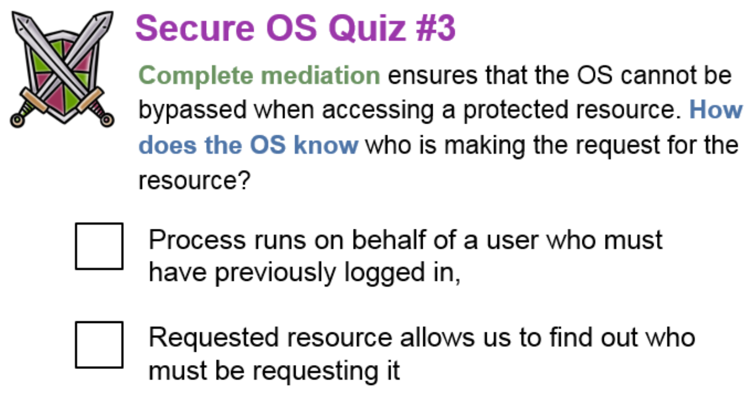
We said the operating system controls access to protected resources. So if the operating system is going to control access to protected resources, and if you are one of those applications that needs this resource, you have to ask the operating system. When you ask the operating system the way you do that is by making a call to the operating system, and that's called a system call. As I said we haven't discussed this, but I want you to sort of think about this question here that says system calls allow application code to gain access to operating system, and particular resources that are controlled by the operating system. A system call is different than a regular call. When you run code, you make function and procedure calls all the time to do control transfer, to go from one part of the program to another part of the program. Here we are going from the user part of the program to the operating system, and when you make a call like that it is special and it is called a protected procedure call. So a protected procedure call is different. And the question that you should think about at this point is how different is it? One way to sort of characterize the difference is the cost of a call. So I want you to think about the cost differences between these two kinds of calls. When the system call is how a user or application code actually goes into the operating system to gain access to some resource it needs, and things like that.



Is the cost of a system call:

- the same as a regular call
- [✓ higher than a regular call]

So the answer to this question actually is the second one. One reason, we're going to get into some details as to how exactly it's different, but you can think about you're doing control transfer. You're going from one part of the program, that's your user part, to the operating system. So you're doing what you would have to do, saving, for example, when you're going to return and things like that. Pass arguments barometers, but you're also switching the so called protection domains. You're going from user space, so user land we call it, to the operating system. The operating system is trusted. It's a distinct protection domain then the application code that we're running. And when you do that, now you're going to be able to do things that you couldn't do before.

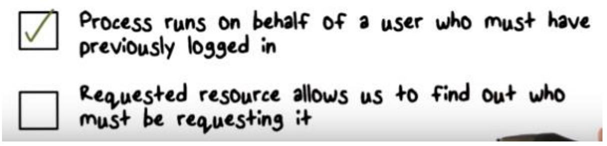


Secure OS Quiz #3

Complete mediation ensures that the OS cannot be bypassed when accessing a protected resource. How does the OS know who is making the request for the resource?

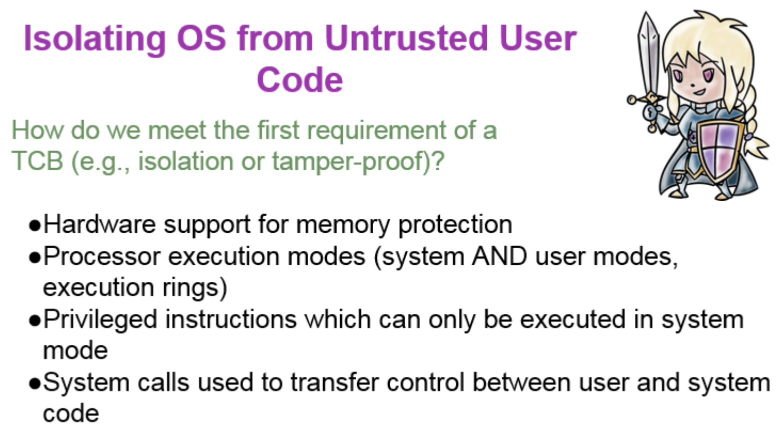
- ☐ Process runs on behalf of a user who must have previously logged in,
- ☐ Requested resource allows us to find out who must be requesting it

Okay. So, one more question which has to deal with one of the requirements we have for a Trusted Computing Base. We know that's complete mediation, which means untrusted code cannot bypass the operating system or the the Trusted Computing Base when they need access to a protected resource. We're going to say that you can't bypass it. And so, you'll have to come to the operating system. And then, the operating system has to figure out who is making the request. How does the operating system know that?



- ☑ Process runs on behalf of a user who must have previously logged in
- ☐ Requested resource allows us to find out who must be requesting it

Well, we know that the Operating System keeps track of applications or processes that we have on the system. Because it has to make various resources available for this processes or applications. So, in addition to if knows on whose behalf a given application is running. And when I say that I really mean. So the user would launched this application. So the answer to this actually is, first one, processes are going to run on behalf of users and that users have to login to the system and launch this application or these processes. System, the Trusted Computing Base as the responsible for complete mediation, is going to keep track of these processes as we said before. So when a process is going to make a request, and I keep saying process, or application interchangeably, an application essentially runs as a process. So when the process makes a request, the system knows on whose behalf that process is running and the user on whose behalf it's running is somebody who must have logged in previously. And after logged in, we had launched its process for that user. The resource that is target of this request, we have to keep track of who all are able to access it. But when we get to Access Control we'll discuss how that is done.



Isolating OS from Untrusted User Code

How do we meet the first requirement of a TCB (e.g., isolation or tamper-proof)?

- Hardware support for memory protection
- Processor execution modes (system AND user modes, execution rings)
- Privileged instructions which can only be executed in system mode
- System calls used to transfer control between user and system code

How is the operating system able to meet the requirements that we have identified for it?

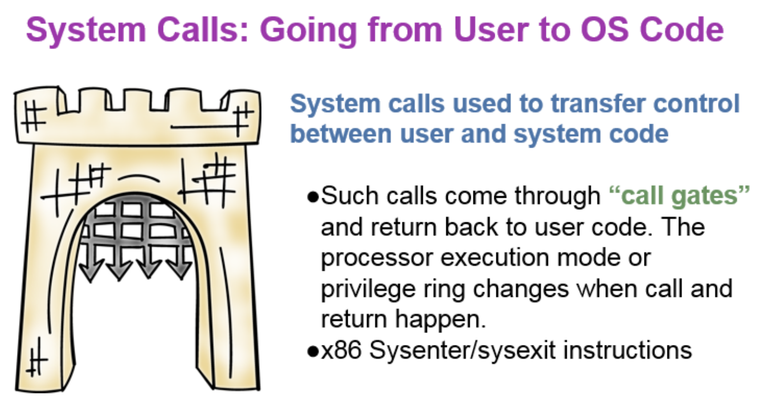
And one of them is this requirement that the trusted part be isolated and not be tampered with by the untrusted application or the user code. For us to be able to do that, we're going to see that we going to need help from the hardware. Okay so, remember operating system layered on top of the hardware that we have. But, the hardware has to implement certain functionality and that has to do with protecting memory where the trusted system is going to reside. And so the hardware is going to help us protect the trusted part of the system from untrusted applications. And that is actually going to be the reason we're going to able to isolate it or make it time for proof.

When you talk about isolating then, we also ever talk about, well something has to be different when you're executing user or application code. Worse is when you actually go into the operating system or when you’re executing their trusted software that makes up the operating system. Well the hardware actually is going to do something else for us that processor that is executing actually is going to execute in different modes. Execution modes say either I am executing user code that is in our trusted, or I am executing system code. So the processor is actually aware of what kind of code it is executing. So one obvious thing we can do when it's executing user code for example the process I'm in user mode. I shouldn't be touching anything that belongs to the operating system or the trusted base that we have. The Trusted Computing Base that we have. When I'm executing in the system mode. Then I'm allowed to access the code and the data that makes up the trusted system. So the hardware is sort of aware of what is going on at a given time. In particular what execution mode is it running in.

The execution mode for this goes by a different name it is also called execution rings. So ring is sort of think about a privilege level in some sense. When you are in the system mode, you have the highest privileges and there could be many of these in fact several processor architectures support more than two. And I'm just talking with user and system. But, the innermost ring, where the Trusted Computing Base resides, the most privileged. As you move out of that, you, for example, could move into the user ring. And then your privileges are reduced, because you can't access what can be accessed by being in the Trusted Computing Base. So this execution mode that we're talking about, whether you're in system mode or user mode, you can also say you are in the ring that you reside in when you are in system mode versus the user ring, or mode. So it goes by different names, either execution modes, system, supervisor, a variety of these names. But the idea here is that the processor for the hardware knows that it's, you know, execution state is such the state indicates the mode that we're talking about. So it's either running with higher level of privileges, it can do certain things that it's only allowed to do when it's running the trusted code, or it's executing the non-trusted or untrusted code when it's in user mode.

So again, memory protection but then this processor being aware of what its execution mode is, that's something else that we're going to need to meet the Trusted Computing Base that we had identified for ourselves.

So, if you are in different modes, we said you should be able to do more things when you are in the system or more privileged mode, compared to when you are in user or less privileged mode. Well, certain hardware instructions, they're called privileged instructions. They can only be executed when the processor is in the inner most ring or the most privileged ring or is in the system mode. Or ring zero it's called. Zero is the most privileged and you go out, zero, one, two, three and become less privileged. So privileged instructions are special instructions that you cannot execute when you are in the user mode. Okay lot of times these instructions sort of help you figure out what parts of memory you access for example. Or these are hardware devices that we have need to control. These kinds of instructions direct access to the hardware we said is not available to user code. Similarly what parts of memory you can access can not be manipulated at user level. If it could be then you can go access somebody else's memory. So those kind of actions require user instructions that can only be executed in the system mode. And these instructions are called privileged instructions or instructions that can be executed in high privileged level, a ring or more that you're in. So one of the ways in which we're going to be able to meet the requirements that we identified for a Trusted Computing Base is by making use of a number of different things the hardware does for us. It has memory protection, it gives us these execution modes we're talking about. It also has special instructions that cannot be executed when we are not in the right mode. So, with this support from the hardware, we will see we will be able to do isolation or the tamper resistance that is one of the requirements for a Trusted Computing Base.



System Calls: Going from User to OS Code

System calls used to transfer control between user and system code

- Such calls come through "call gates" and return back to user code. The processor execution mode or privilege ring changes when call and return happen.
- x86 Sysenter/sysexit instructions

Okay so we already talked about system calls. In fact, I said think about the cost of a system call compared to a regular call. So system calls allow you to go from user mode to OS or the system mode. So let's talk a little bit more about system calls.

First of all, they're used to transfer control between user and system code. Or execution that must happen in user mode and system mode. The interesting thing about these calls is that they cannot be arbitrary. These are a set of calls that user level code is allowed to make. Okay, this is defined by the application programming interface, or the API that your operating system provides. So, you have these calls, and these calls have to come into the operating system in a control fashion, okay. So we said these calls are different than regular calls. They have to come through what used to be called "call gates", which is special ways in which you can enter from user to, it's a defined way, transition from user to system level. You come through these "call gates". And what happens as a result of that is, well we know that it's a call, so we going to return at some point. So we have to keep track of where we going to return and the user code once the system call completes. But the processor execution mode has to change as a result of that. We went from user mode to system mode so the privilege ring or the mode is going to change. And we actually going to be able to, we'll have to change some memory mappings, data structures that keep track of those, because we going to be able to access memory now that we couldn't access before.

So some registers have to be saved. Others have to be loaded. And things like that. So all that work has to be done. So now we are executing in a different protection domain. In fact, this used to happen through, sort of an interrupt or a trap into the system from user mode.

But in x86, we actually have explicit instructions. This is different from your regular call return. These are execute a system call. That “sysenter” the system or the operating system and when you're done you return by doing another special instruction called “sysexit” so there are these instruction that help you implement this system calls that we're talking about. Keep in mind that they are more expensive because now we're using, we're doing this work that we didn't have to. We have to check what kind of call it is, arguments that are being passed. Same information when we come back. We change memory mapping so we can access things that we couldn't before, use the special instructions rather than the regular ones and so on. So that's what makes it costlier or more expensive.

Isolating User Processes from Each Other

How do we meet the user/user isolation and separation?

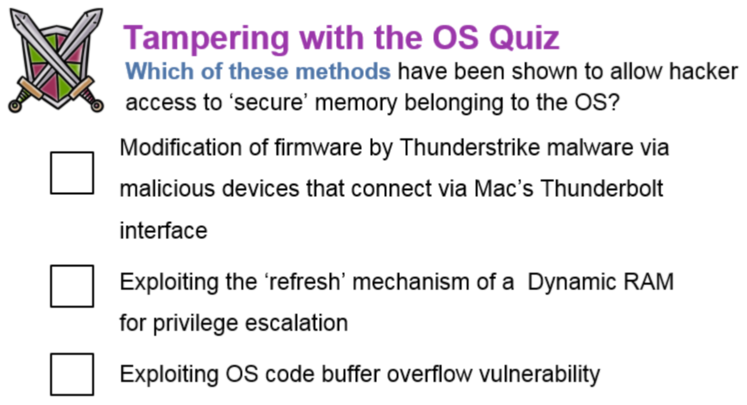
OS uses hardware support for memory protection to ensure this.

Okay, so let's revisit this idea of isolation. When it comes to protection and security, isolation is actually your best friend. Okay, that means that I'm isolated from somebody else and they can't do anything bad to me. Untrusted user code has to be isolated from the system code, okay.

We talked about trusted system code tamper-proofness that we're talking. You shouldn't be able to alter it and the only way you can go in to operating system is through system calls that are defined by the operating system and so on.

So how are we going to achieve this isolation of user code? The reason I wanted to think about this is what does a processor do? It says fetch the next instruction. To execute it you need some operands that are somewhere in memory. You fetch those you execute the instruction and you keep doing that forever. But I'm using let's say running executing some user code. Why can't I say the next instruction I want to switch to somewhere in the operating system or the next data item I want to bring is a data structure that resided in the operating system? If you tried to do that we said you wouldn't be able to do it. Because we have this execution mode we're talking about, and things like that. So, we're going to use that hardware support to achieve or accomplish this isolation, separating, or separation, of user code from the operating system code. So let's see how we actually going to be able to do that.

Well the way to do that is again I said. It's going to rely on the hardware to protect memory. So remember isolation I said the way. Executing in user mode or executing user code and at that point if the hardware can protect memory where the operating system is. What does memory protection mean? We said the process in which read write request or load store request or execute those instructions to fetch the next instruction or fetch the data that it's going to operate on. It's going to generate an address where it wants to do a read or write. Well, the hardware says, you're not allowed to access that part of the memory. Okay, hardware support is saying, it is part of memory that belongs to the operating system. Okay, that's what the operating system code and data is. If you're running in user mode, you are not allowed to generate an address and complete a read write in a memory location that belongs to the operating system. So hardware support memory protection essentially says, if the process happens to generate an address that is in the operating system, the hardware support we have for memory protection is going to stop that memory access from completing. Although it's generating an address, and it's a memory location, the hardware that potentially can do a read or write there but it currently knows that it's user code. And user code directly can't access the information that we have in the operating system part. So the hardware is going to essentially stop that access from proceeding. So hardware is going to sort of keep you contained in your space. And we get to that, it's not just OS user, but it's also across different user processes, but it's going to contain you within the area of memory that belongs to you. And if you try to touch something else. Which you can. It's going to stop you from doing that. And the hardware is going to do that. All this hardware support we have for memory protection is actually going to do that.

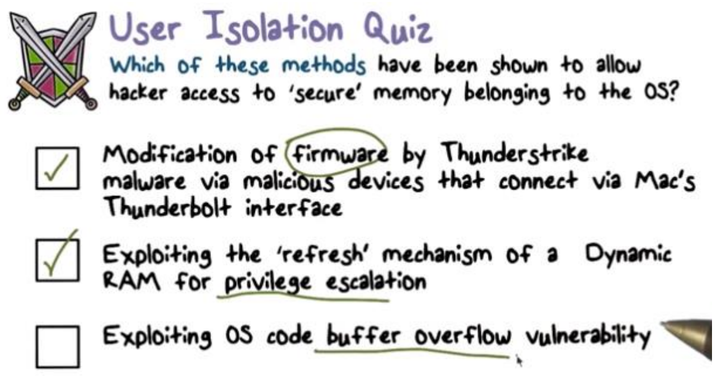


Tampering with the OS Quiz

Which of these methods have been shown to allow hacker access to 'secure' memory belonging to the OS?

- ☐ Modification of firmware by Thunderstrike malware via malicious devices that connect via Mac's Thunderbolt interface
- ☐ Exploiting the 'refresh' mechanism of a Dynamic RAM for privilege escalation
- ☐ Exploiting OS code buffer overflow vulnerability

So now that we have talked about the importance of isolation of user code and the hardware support with the process execution mode, memory protection, and so on. I think it should be clear that if you're the bad guy, and I did say that the operating system has the keys to the kingdom, the best thing you can do is actually compromise the operating system. We said well if using user code, it's hard for you to do that, because if you tried to touch something that belongs to the operating system, the hardware is going to stop you. So attackers actually have been really creative, because the operating system of the Trusted Computing Base being such a big target, they have been really creative in terms of how can they gain access to the operating system and the information that it keeps or maintains to control access to resources. Now, I want you to explore these three different options that we have here and then we'll talk a little bit about them.



User Isolation Quiz

Which of these methods have been shown to allow hacker access to 'secure' memory belonging to the OS?

- ✓ Modification of firmware by Thunderstrike malware via malicious devices that connect via Mac's Thunderbolt interface
- ✓ Exploiting the 'refresh' mechanism of a Dynamic RAM for privilege escalation
- ☐ Exploiting OS code buffer overflow vulnerability

So one thing, if your attacked, you can think about is, how can I actually get into this process when the operating system itself is coming to life. And the operating system, data structures are getting set up and information is getting initialized. How can I do some mischief at that point.

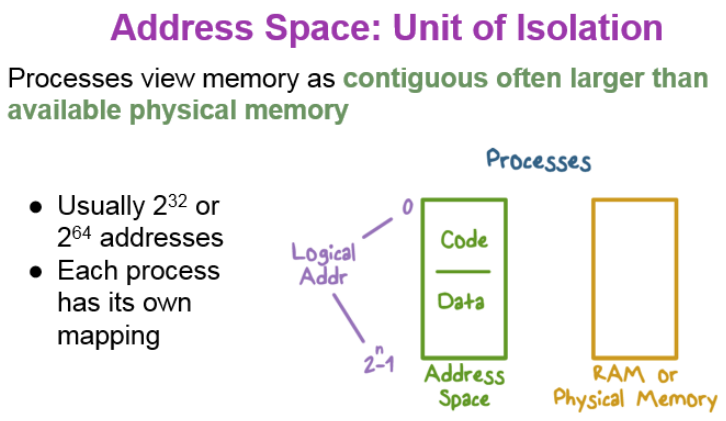
Instructor Notes

Googler's Epic Hack Exploits How Memory Leaks Electricity
Hacking MicroSD Cards

I know once I start running user code, once the operating system is in place, the hardware is going to stop me from going to the areas where operating system cordoned data is, unless I do a system call. But can I get onto it in the early process, it's sort of what you would call infant mortality kind of thing. Attack it when it is coming to life. So the way to think about it is that when you actually power-on a system, a computer system, it runs a set of instructions. And those instructions are what is called firmware. So these firmware instructions actually run and the execution of those instructions leads to finding where the operating system is, where it goes, loading it and setting it up. So, I said get onto it early in the process. Well, there was an attack like that, which was demonstrated against Mac OS. Macs have this Thunderbolt interface. So what you can do is you can have a device that you have maliciously prepared that device, you can connect it through the Thunderbolt interface, the very early stage when the Mac is booting. So this is called bootware or Malware that you can inject at the boot stage. So, because the Macs have this extensible thumbware framework, you can inject new thumbware by connecting this device to the Thunderbolt interface that we were talking about. And you can change the firmware, when it's running. You can actually run this malicious instruction from the malware that your malicious device has that you connected to the Thunderbolt interface. So there was an attack like this actually we are going to provide a link where you can go read more about it. And interesting thing about this attack was that there's no protection against it.

The second one is actually really I said people get creative to go after the system. And this is another one that really exploited some property of the dynamic RAM that we have. So we talked about memory, the DRAMs, dynamic RAM, and the attack actually exploited, if you read some locations, the RAM locations, the RAM addresses repeatedly then the way it is refreshed and the charges that hold the value that stored in that location work. Actually this was demonstrated that by reading aggressively, reading again and again in certain locations you can actually flip some bits in an adjacent location. Okay if the bit flip happens to be the right way, in particular in a location that is storing information protection and information about memory, then you can flip it and make something that will be readable, you can also make it writable. We're going to talk about these protection bits, that you can only read certain parts of memory, but you can write also. So this attack, again, I want you to read and maybe the details. Here is not important. But sort of think about it by exploiting some vulnerability that is DRAM. The way these DRAMs are designed, someone is able to gain access to the OS part of the memory and change something just by doing reads. The operating system doesn't stop you because you just reading certain locations. But those reads actually have this other effect. In particular, this undesirable effect where they end up changing a bit, flipping a bit in another location in the operating system. Which, sort of, again, I should say that this is not easy to pull off you can obviously corrupt or affect integrity of that location but to change it exactly the way you want to do this idea of again privileges that you didn't have. For is quite challenging. Again this is reported attack.

If your OS is written in low level language so it performs reasonable whatever it is. Typically not type safe languages. And if you're not careful you can have buffer overflow possibilities too. Buffer overflow means you can alter the return address. Maybe you can insert some code that you can transfer control to and that is going to lead to exploitation of the operating system. And you're in the operating system running your code, that hacker's code, and you can do whatever changes you want to do. Of course the isolation is lost in that case. So there are instances of attacks like this too.



Address Space: Unit of Isolation

Processes view memory as **contiguous often larger than available physical memory**

- Usually 2^{32} or 2^{64} addresses
- Each process has its own mapping

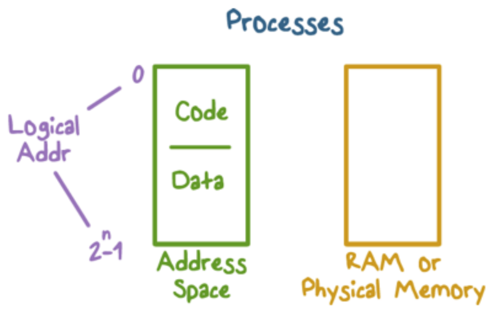
Okay, so I talked about memory, I talked about isolation. Transferring control from user to OS part of the system that we have, a memory where OS coordinator structure is. We want to sort of dig deeper into it and really think about how do we define what data and code a certain process or application can access.

I said one of the jobs the operating system has is give you high level abstractions. Okay, so we don't talked about particular memory location that we happen to have right now. With each process we said sort of can believe as if it has the whole computer to itself. In particular, each process wants to feel that it has the memory that it needs, where its data and code is going to go, it has to itself. And actually, each process does have something of that sort for itself, and that's called an Address Space. Address Space is sort of a container. It's a sequence of memory locations. It's a collection, which is a sequence of memory locations that define a collection of those memory locations, it defines a space, and each of this location can be address so that's what makes it an address space.

Because each process has an abstraction of memory that is an Address Space, an Address Space actually is going to be a unit of isolation, so when we talked about isolation. Now we're talking about sort of isolate what from what else, and we're going to talk about isolating Address Space. Address Space completely defines the data and code that is there for a given application. The data and code is going to reside in the bunch of memory locations and those memory locations together make up an Address Space that belongs to a particular process or application. So that's why we have to sort of talk about the topic of isolation. We have to dig deeper a little bit and think about the Address Spaces.

So if a processes essentially going to think that it has the memory to itself, it's going to think of this memory as a contiguous set of locations going from some starting address zero to some max. And it's going to think that it has this set of memory locations, not only the contiguous, but they could even be more than the total memory that we have, physical memory that we have in a computer. If you done your operating systems class, you heard about virtual memory, which most operating systems actually support virtual memory. That could be more than the physical memory that we have. Address Space that we're talking about, when a process is a contiguous container of memory, okay? That can be addressed, which is going from zero to some maximum address that we can have and it doesn't have to be constrained by the physical amount of physical memory that we have. So the address space actually usually, it it's a 32-bit architecture, basically means addresses in that case are 32-bit long. Or it could be 64-bit architecture. In that case they'll be 64-bit long. They go typically form zero through two to the third minus to minus one, or two to the 64 minus one. Your address is “n” number bits, and based on that you can have two to the “n” different locations that you can address with that size of an address.

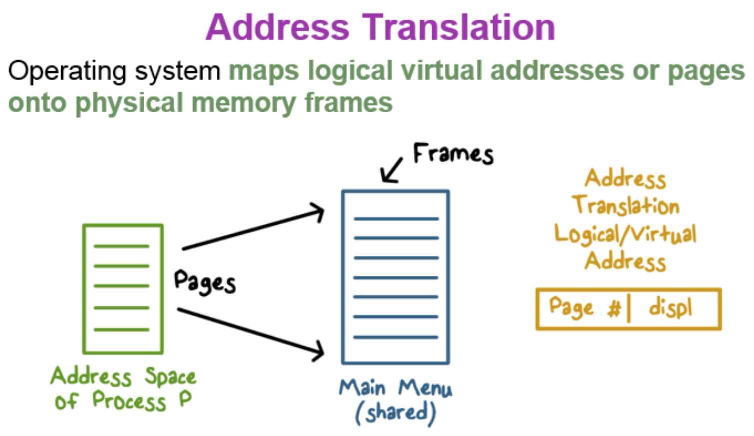
We said that Address Space is a contender for anything that has to be stored in memory for a process. Remember, process has to fetch instruction data, so that information has to be in memory before we can execute it. So we have to prepare an Address Space for a process before it starts execution. And we tell the process, here is an Address Space for you, you can address the location zero through something, and you decide to put your data and code in these locations.



Okay, so let’s look at what happens, I call this Logical Address process. So we always talk about physical addresses, those are addresses that point to locations in physical memory. Locations in address space are logical its a logical or virtual address because it's going to map to, okay so this mapping that we're talking about here, it's going to map to some physical memory location. Eventually everything has to be stored in physical memory. So we can say logical address 2000 in process one's address space is currently lives in physical location 5000 or something like that. So process the address space that we're talking about and the addresses that make up that address space, zero through I said two to the n minus one, n is 32 or 64, these are called logical addresses. This is in these logical addresses, or the logical space that we define for a process. The process is going to put it's code, and it's going to put it's data.

So, when we talk about an address space as a unit of isolation, so that if you start with this process had its own container of memory where it can address different parts of it in particular different locations where it's going to place its code and data, and these are logical, or in case of virtual memory, virtual addresses. Eventually, remember the operating system has to use physical resources to implement these abstractions of virtual resources. Well address space is a virtual resource that abstracts memory. So this address space or whatever that we have here, the information that we stored in this address space, eventually we have to use physical memory, okay, that's RAM or physical memory, to actually implement this abstraction we call an address space. So you enter the logical address, you say I want to access logical location 2000. We have to say well that currently lives somewhere in this physical memory we have here. So this code and data eventually has to be placed in different places in the memory that we have. This address space has to be backed up, physically has to be supported with the physical memory that we have. And this physical memory could be multiple processes having their address spaces concurrently being or existing in this physical memory that we have.

So one of the things, isolation we're talking about is that if I'm giving up some physical memory for address space to do process A, and I give some physical memory to another process B, it then uses that memory to store some of its stuff from its address space. I have to isolate the two address spaces, I actually have to isolate both physical memory can process A access versus what physical memory can process B access. So the operating system has to do that. I think isolation, the way to think about this is when process A is running, it's adding space is in some parts of this physical memory, it should only be able to go to those parts. When process B is running, it should be able to go to only those parts that are currently allocated to it. And they never run into each other unless they choose to share, which is always sort of an exception. So when you talk about isolation, think of an address space that's what a process thinks it has, implementing that address space by storing pieces of pieces of indifferent parts of memory, and then making sure that the process can only go to physical memory that belongs to it. And that's how we get this isolation. So one of the things we want to do is how exactly does it work, so to get some sense of that.


Address Translation
Operating system maps logical virtual addresses or pages onto physical memory frames
Frames
Pages
Address Space of Process P
Main Menu (shared)
Address Translation Logical/Virtual Address
Page # | displ

Address translation process. And the reason that translation has to happen is because process thinks it has this big address space, contiguous going from zero through some x. And then we said, well the physical memory, one process may not have all of it at the same time. Or physical memory maybe much smaller than the address space size itself. So logical address or virtual address is actually going to be different from the physical address that we're talking about. So this is actually, if you think about it, this is a logical address here. And addresses that we have here are going to be addresses into physical memory. Right, that's your real memory that you have in the system. So what we're going to do is, we're saying, well, this part is going here. That part is going here. So, maybe 1 goes at the top here, maybe 3 goes down here, and things like that. We said we have to take data and code that is in the address space, and we have to place that into physical memory, okay? The way we sort of do that is, we don't want to do it for each byte or each word, because we'll have to keep track of what's going where.

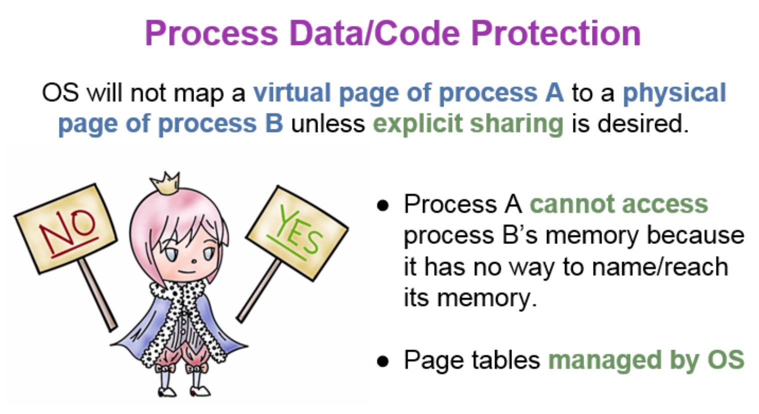
Okay, so one way to scale that process is is that we divide this address space and the memory into some size chunks. Okay, so if we do paging, it's divided into what's called pages. A page typically may be 4 kilobytes, for example, 4K size. So we would say, well, the address space really is a page number and displacement within that page. So this is page 0, and if your location 10, then it's page 0, 10, okay. So 0 is the page, and 10 is displacement within that page. If your address happens to be let's say, 4K plus 3, then it's going to do a second page because second page starts at 4K. First 0 through 4K minus one goes in first page, and then the second one starts. So then we'll say it's page 2 and maybe displacement 5 or 2 or whatever it is. So the logical address you can think of that as a page number and a displacement. Essentially a page table says, this, it is a mapping isn't it? So it says this logical page is currently stored in this physical page. Okay, so if I want to think about it, a page table, essentially, a page table we have all these entries. And it's logical base 0, 1, 2. We're saying this may be a 3. This may be at 5. This may be at 8. And maybe 3 is currently not loaded. Virtual memory, not everything is in memory at the same time.

So this a page table that actually we build, and the operating system is responsible for actually building this and protecting this.

Remember, at this point, sort of a flow that we should understand, it's really important. A process, maybe executing in user space, generates an address, okay, saying I need to fetch the next instruction. That's where my instruction point or program counter points, or an operand, or whatever it is. And I need to access this memory location. It sort of starts with a logical address. The system now says, well, where exactly is this information that the process needs? We need to locate the physical address to which this logical address maps. For that, what I'm going to do is I'm going to take the logical address page #, and I'm going to look up this page table. So this address translation requires that you access this page table, and there are ways to speed this whole process up. So it's saying if page number is 3, or let's say the page number is 2, then I go down here and I say oh, it's in physical memory page 8. So the address that we have, if it's 3 and some displacement, d, could translate to 8 and the same displacement. So it's say memory page 8 and go dig down d locations or d bytes in that. This is what is called the address translation process. So an important observation here is that we can only access physical memory that is reachable through this page table that we have for this process.

I should say page table a lot more complicated than multilevel, things like that. But let's just sort of stay with this basic idea of how they help us get this isolation which is important to us because the process has to start with an address. Address has to be translated. Translation is only possible through this page table. So whatever the page table points to is all a process can access.

Another observation here that we should be clearly understanding, is, the page tables have to be maintained by the operating system. They have to be managed by their Trusted Computing Base because whatever goes into the page table, we can access that. So the operating system should make sure that whatever goes in the page table of a particular process is only memory that belongs to that process. So, that's the address translation. It gives you a little bit more information about what part of memory a given process is able to access, and it relates to this isolation thing that we've been talking about.



Process Data/Code Protection

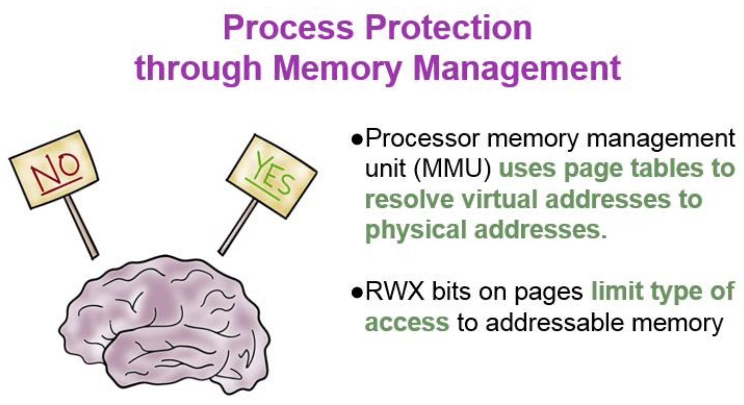
OS will not map a virtual page of process A to a physical page of process B unless explicit sharing is desired.

- Process A cannot access process B's memory because it has no way to name/reach its memory.
- Page tables managed by OS

So this is this address translation, and what you can access in memory is really powerful. This is how we're going to be able to protect the data and code of a given process. And the way we're going to be able to do it is, the operating system, I said is responsible for managing those page tables you're talking about.

Remember you always start with a virtual address. So process is going to start with virtual address, which has to be translated to physical address that should fall in a physical page that we're talking about, physical frame. So the operating system will not map a virtual page of a process A, but remember you're going to start with a virtual address that has a virtual page number and displacement within that. It will not map a virtual page of process A to a physical page that has been given to process B. Remember these pages are reusable resources or time to get assigned to different processes so that currently a certain physical page has been assigned to process B. The operating system is not going to set up the page table of process A, such that a page table entry in that process A's page table, can point to the space that belongs to B. That's our protection isolation whatever you call it. We have essentially partitioned the physical memory that we have, among the different processes we have in the system. And we limit these processes to only the portion of memory they have access toward a given time. By making sure that they can only reach those portions through their phase tables, and the operating system is again going to manage their phase tables. The Trusted Computing Base you trusted to do this right, so we keep a different processes separate from each other. Which is what results in getting protected from each other and process A doesn't trust process B.

So as I explained process A cannot access process B's memory because it has no way to read the memory that belongs to process B. This will work as long as this page table is correctly managed. And as I said before they're managed by the operating system which is trusted so let's hope they are correctly managed unless somebody finds a way to attack and successfully compromise the operating system.



Process Protection through Memory Management

- Processor memory management unit (MMU) uses page tables to resolve virtual addresses to physical addresses.
- RWX bits on pages limit type of access to addressable memory

So protecting processes from each other and protecting the operating system from untrusted process code, same exact mechanism. The page table, when you're running in user mode, doesn't allow you to go to any physical pages that belong to the operating system. It's really done through this memory management that we just talked about.

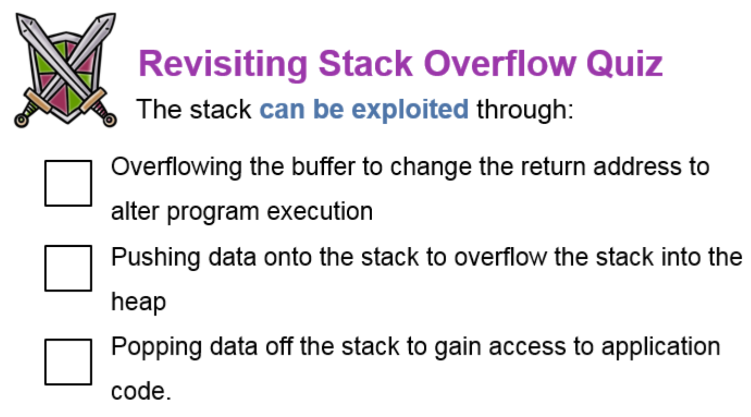
The memory management is important enough that we actually hardware wired support for it. Typically, you have what is called a memory management unit, or MMU, that helps you take those logical or virtual addresses and resolve those into physical addresses efficiently. Except we had to go through the page table then to access memory location, you have to access page table entry and that adds overhead.

So the hardware actually can store some of that in things called TLBs, or translation lookaside buffers that are associated memory that allows you to do this mapping fairly efficiently. Memory management is translation, and isolation, what you can access, and all that, there's hardware support for it.

So there's another interesting thing that we should be able to think about here. So we have these units we call pages or frames, and we have this table we call a page table. And we said we go through a page table entry to see where our logical page currently lives in physical memory. So that page table entry tells us where in physical memories. But we can have some other information along with that. We can, for example, have whether this page that we're trying to access is for reading only, is it for writing, or it's for execution only in case it happens to contain instructions or a code. So you can actually have this kind of protection information on these pages, in the page table entry that can limit the type of memory access you can do.

So we said protection sort of, now we are saying there are two parts to it. First part is, where in memory a given process is allowed to go. So where can its logical addresses be translated to in physical memory.

But now we are saying another level of protection is, in what manner can it access, even if it's allowed to access a given location in physical memory. Is it reading only? Is it writing only? Is it execution and things like that? So the way we do that is in the page table entries we keep this information, this protection information about the ways in which memory can be accessed. And that is also used when you're trying to do, let's say store, and the target address is write protected. You can't do that - that instruction would not be allowed to go through by the hardware.

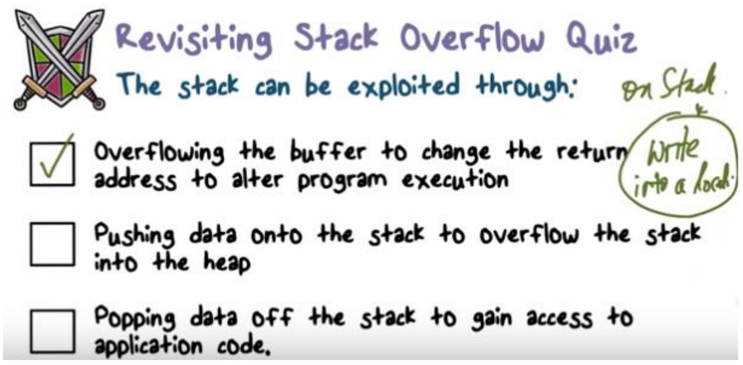


Revisiting Stack Overflow Quiz

The stack can be exploited through:

- ☐ Overflowing the buffer to change the return address to alter program execution
- ☐ Pushing data onto the stack to overflow the stack into the heap
- ☐ Popping data off the stack to gain access to application code.

Well all this talk about memory protection and pages, and read write execution information, and things like that. Probably a good point when we can revisit this Stack Overflow quiz we did when we talked about software security. So Stack and we exploited in a variety of ways. So the first question here in this quiz, is go back and think through how we could exploit the Stack if we don't have memory for protection. And then we're going to come back and see how memory protection mechanisms that we just discussed can actually help you stop that kind of exploit. So the first one is just how do you exploit the Stack.



Revisiting Stack Overflow Quiz

The stack can be exploited through:

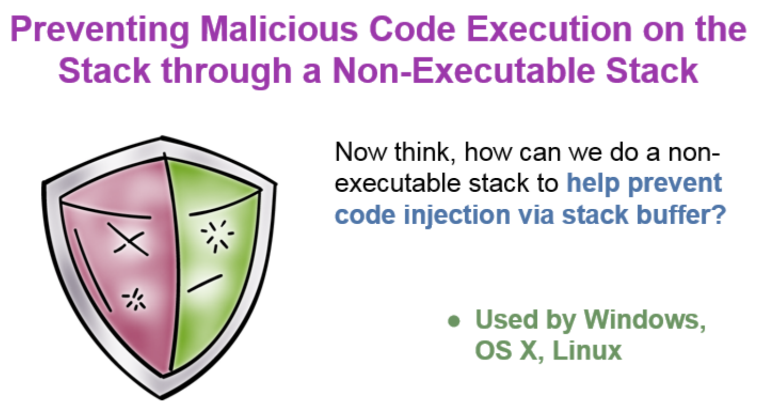
- ✓ Overflowing the buffer to change the return address to alter program execution
- ☐ Pushing data onto the stack to overflow the stack into the heap
- ☐ Popping data off the stack to gain access to application code.

So remember what happened when we're talking about smashing the stack for fun and profit, you overflow the buffer and go down where the return address is stored and then you change that if you going to alter the program execution. So actually had to do two things, you had to change the return address and then you had to send the return should happen to your shellcode where the exploit code is. So you overwrite that, the return address, so this is going to be true. It's correct.

The second says pushing data onto the stack to overflow the stack into the heap. The heap and stack are sort of separated, the buffer overflow, although the heap overflows and stack overflows, but going from stack to heap is not what we discussed, so that's not correct.

And popping data off the stack doesn't do anything for you either, because you need to actually alter the return address and go to exploit code. Neither of these two, second and third options are going to let you do that.

So, the way to think about this is overflowing the buffer to change the return address. So essentially what you're doing now write into a location. So keep that in mind. Where is this location? On the stack.



Preventing Malicious Code Execution on the Stack through a Non-Executable Stack

Now think, how can we do a non-executable stack to **help prevent code injection via stack buffer?**

- **Used by Windows, OS X, Linux**

So all the memory protection stuff that we have talked about, how can we use that to prevent malicious code execution on the stack?

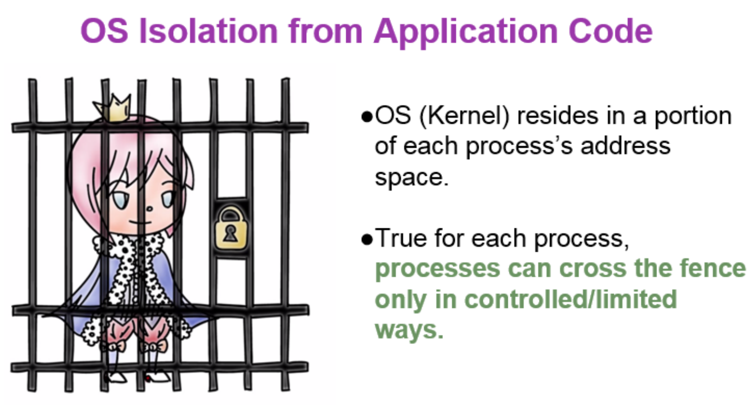
Okay, remember, you're going to alter the return address, then you're going to insert some instructions where you're going to transfer control and things like that.

So one of the things we're going to do is, what if we have a non-executable stack? We actually talked about that's one of the protections for code injection, because you have to inject your code when you overwrite the return address to point to this code. And then you execute that code, that's how a successful exploit works.

So now think what we can do if we made the stack non-executable, right? So this is important, non-executable stack, which means you could go right into it, you can store some instructions into it. So in some sense you're able to inject the instructions onto the stack, but you will not be able to execute those, okay? Because remember we're talking with address translation and space level entries, and read write execute permissions in each of those pages that you can access through the space level entries. Well, if we say it's non-executable, non-executable is you can only read or write to those location when you want to fetch an instruction off the stack. This is your injection code that we are talking about. You can't execute off of that, so setting the permission to non-execute actually is going to prevent malicious code to inject code on the stack, and use the buffer overflows to transfer control to it, and successfully exploit a program by doing that. A simple thing we can do is, you can't execute off the stack, so your injection code can't be on the stack.

So, this sort of protection, actually more than operating systems, make the stack non-executable. So, the way you understand that is, this protection comes because that portion of the address space where the stack lives. The page table entries that are used to translate addresses from the range of addresses where the stack is, those page table entries have execution permission turned off. So you can't execute off the stack because it's going to require that you fetch an instruction from a certain memory location where the execute permission is not there.

So memory isolation protection, that discussion we've been having, gives us a simple way to protect our execution on the stack by making the stack non-executable. And that avoids or prevents this malicious code to use a buffer of overflows and use code injection on this stack.



OS Isolation from Application Code

- OS (Kernel) resides in a portion of each process's address space.
- True for each process, **processes can cross the fence only in controlled/limited ways.**

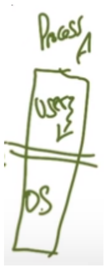
Okay, so one nice thing about the memory protection stuff we've been talking about is that it could be used to create these fences or isolate these different applications from each other.

I was talking about process A cannot access memory that currently belongs to process B. But the same mechanism actually could be used to isolate the operating system.

Remember, we wanted to tamper resistance for the operating system. You shouldn't be able to alter from untrusted application code, so we can isolate the operating system from untrusted application code using the same memory protection mechanisms that we were talking about.

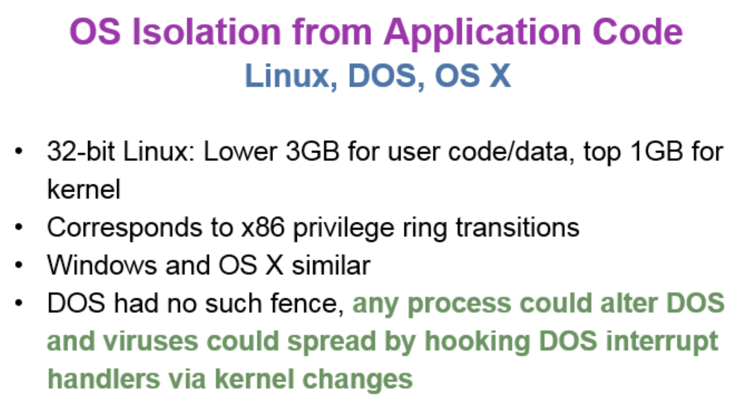
So the way we do that is we're going to say the operating system, also called the kernel, is going to reside in a portion of each processes address space. So the address space in which the process executes now has two parts. Where the user code and data is going to go, and where the operating system is going to go, operating system code and data. So the OS resides, and that part of the address space is common across different processes, is going to be a portion of the address space. So think about the address space. There's a user portion, a user area, and then there's the OS, or kernel, area.

And this holds for each and every process that we have in the system. And remember that we talked about system calls and things like that. Whenever you want to go from the user portion of the address space to the system portion of the address space, that transfer has to be special and handled by the operating system. It can only be done through the system call that we're talking about.



So one way I want you to think about it is that, but so this is the address space that process A executes in. Then we said there is this fence or boundary. The OS goes here, the user code goes here. And when you are here, when you're executing this portion, you're not allowed to access this part of the memory. Remember the execution mode that we're talking about? So the hardware is going to save you in the user mode, and you're asking for an address here, that's not going to be allowed. So the way all operating system is isolated from application code, which is untrusted, is essentially putting those in separate places, and making sure when you're here, you can't go into the operating system. Well, we said that call gates are controlled entry points through which you can come in, but then we're coming in by making a certain a call, the operating system understands, and the operating system is going to do whatever checking it needs to do. And then you're going to be executing in the operating system while you access anything here. And when you're done, you're going to return back to the user level.

So we use the same memory protection mechanisms you're talking about, base tables, protection bits, what different base tables we have for different processes for the operating system. So, remember, one of the things that I can't overemphasize is the operating system, or the trusted computer base that we have, has to manage these page tables because page tables control what portions of memory you can access, and in what manner. And the operating systems says, this is your code, you can execute it, but you can only execute. Or this is your stack, you can read/write it, but you can't execute from it, that's a non-executable stack, and things like that. So this is how the operating system is going to actually protect itself from untrusted process codes, and also allows these untrusted processes protection from each other. So process A can't, for example, corrupt the memory of Process B because the way the address translation and the protection mechanisms work.



OS Isolation from Application Code

Linux, DOS, OS X

- 32-bit Linux: Lower 3GB for user code/data, top 1GB for kernel
- Corresponds to x86 privilege ring transitions
- Windows and OS X similar
- DOS had no such fence, **any process could alter DOS and viruses could spread by hooking DOS interrupt handlers via kernel changes**

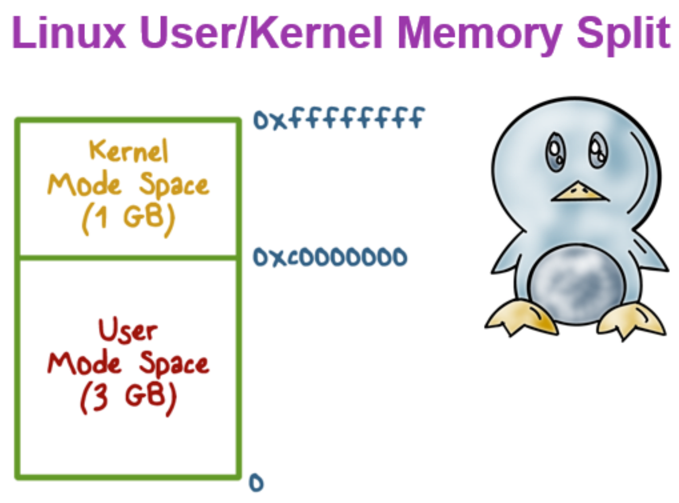
Let's consider a more concrete case. Consider some of the operating systems we currently have or had in case of MS-DOS and Vanguard. So what exactly I said in the OS portion and the user portion of the address space, how does it work in these systems?

So for example, if you look at the address space layout for 32-bit Linux, lower three gigabytes is where the user code and data is going to go. So address space is going to be four gigabytes long, with 32 bits. So we're going to separate that and say one gigabyte for the kernel and three gigabytes for the user. The lower three gigabytes, that's where the user code/data is. The top one gigabyte is where the kernel goes, and there's this fence between these two. The kernel portion can only be accessed when you are in the operating system or in the kernel.

Another thing to keep in mind is that these portions of address space that we have, access to those address spaces also is related to what execution mode or privilege ring you are in, so x86 has the zero through three. For you to be able to access the kernel portion, you have to be entering zero or in system mode if you are executing user code in ring three, you can't access this portion of the address space that we have, which is the one gigabyte. So this connects back the hardware support that we have and the address space like we have been talking about.

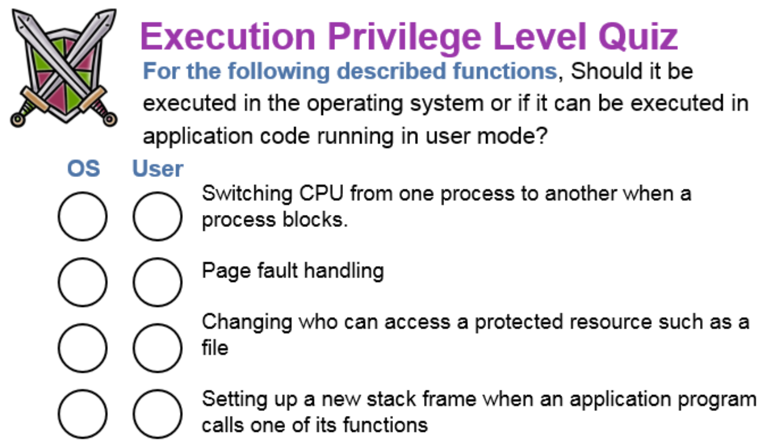
So these other operating systems, depending on what particular version, may have some differences, but typically this is how it works. The operating system is common, so this kernel one gigabyte has the same data and code for every process, but the three gigabyte is for process, obviously. This could be different for different processes. And we said, if you're executing this part of this three gigabyte, then you are in ring three, if you're executing in the kernel, you are in ring zero. This how operating systems manage access to various part of the address space.

So modern operating systems actually separate the Trusted Computing Base from untrusted. It wasn't always like that. If you look at MS-DOS, and actually now it's in a museum, Microsoft even, the first version they made before is called Available, actually, written in assembly language, it didn't have the separation that we're talking about. There's no sort of fence or wall between the user and system code, so any process could actually alter the operating system, because you can freely march down into the OS space. And if you happen to be virus, you obviously could hook anything, including here, DOS interrupt handlers actually are always in the operating system. We are talking about dealing with hardware device, and direct access to hardware it through privilege instructions, not from user level, those instructions have to be privileged. Though, here you could actually change the kernel, change the interrupt handlers, there was no protection that modern operating systems provide through this isolation of the operating system from application code.



Linux User/Kernel Memory Split

So this is just, in picture form, the address space 0 to 4 GB. This is the mode 3 GB we're talking about, which is the user space, user core and data and it can be accessed in user mode, or the ring that corresponds to user mode execution. And the high 1 GB has to be kernel mode, and that's where the kernel data, and kernel code goes. So, this is just the split that we're talking about here showing, drawing the address space starting at 0 going to the max, which is to the 31 minus 1.

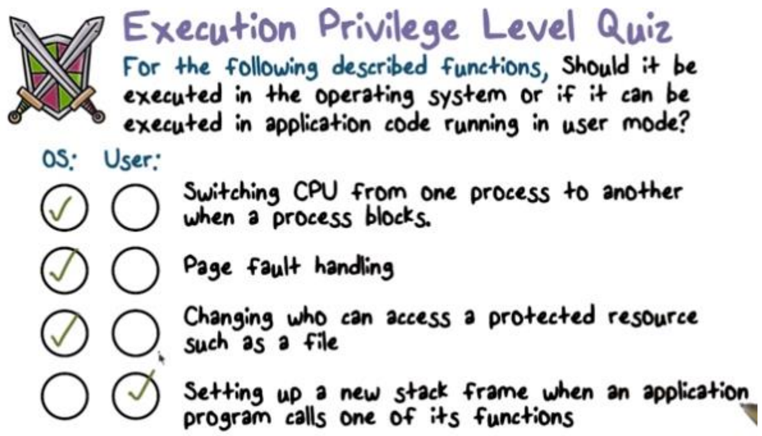


Execution Privilege Level Quiz

For the following described functions, Should it be executed in the operating system or if it can be executed in application code running in user mode?

OS	User	
○	○	Switching CPU from one process to another when a process blocks.
○	○	Page fault handling
○	○	Changing who can access a protected resource such as a file
○	○	Setting up a new stack frame when an application program calls one of its functions

Okay, so we're ready for a quiz now. We talked about sort of system mode and user mode or previous level that is needed when you're in the kernel, and user mode. So in this quiz we're going to talk about a couple of different functions. These functions could be executed either in the operating system, so you have to figure out if this function belongs in the operating system, or it could be executed in application code that is running outside of the operating system, and that would be in user mode. Depending on what this function is, if it can only be carried out in the operating system, which is kernel mode, then you should check operating system. If you think it doesn't belong in the operating system and can be done at the user level, then you should check user. Okay, so there are four different functions in this quiz that you are to think about. And for each, decide if it's OS, system mode, or user mode. Goes in that system portion of the address space, one we provide at the top, or can it be executed in the user part of the address space?



Execution Privilege Level Quiz

For the following described functions, Should it be executed in the operating system or if it can be executed in application code running in user mode?

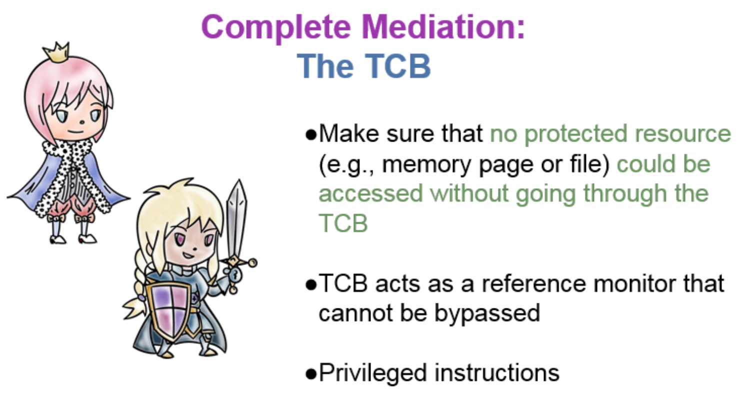
OS:	User:	
✓		Switching CPU from one process to another when a process blocks.
✓		Page fault handling
✓		Changing who can access a protected resource such as a file
	✓	Setting up a new stack frame when an application program calls one of its functions

Okay. So the first one is switching CPU from one process to another when a process blocks. When you go from one process to another process the address spaces are going to change. Yes. The address spaces are going to change because the incoming process now is going to use its own page table to access its data and code. Where can we change address spaces? Remember as soon as you change address space, you changing what portions of physical memory can be accessed by whoever is executing. Address space change always has to be in the operating system because that's how we protect memory. Okay the operating system manages what page table currently can be used by the process that has the CPU and is currently executing and so on. So that has to go in the operating system.

Page fault handling we didn't talk about virtual memory that page fault handling comes in virtual memory. So if some virtual page is not currently in physical memory then the page table entry is going to say it's not in memory. It's not a valid frame that this logical or virtual page maps to. In that case, we have what is called a page fault. We have to bring this from disc, or swap area, or whatever it is. Find some free space in memory, and then update the page table of this process to point to this new page in physical memory where this information was just brought in. So if you update page tables, again you have to be in the operating system. If you could update a page table from user space than you can update with a different frame number or physical memory, some other place. And then you can go there, even if it doesn't belong to you. So in user mode you should never be able to update page tables, and that's why it belongs in the operating system.

The third question we have, in changing who can access a protected resource such as a file? Changing permissions. I can't do that at the user level. If I can change permissions for files that belong to a different user, Alice can do this for Bob's file. Well, she could go access Bob's file even if Bob doesn't want her to access it. The way we should do it is, go to the operating system, and request that can we have access to it? If Bob has done something that allows Alice to gain this access. That's fine, otherwise the operating system is not going to add this permission for Alice to the file. So, this is another function protected resource accessed to it, the operating system should do it.

Last one, setting up a new stack frame when an application program calls one of its functions. We know that happens at the user level. If we call functions back and forth, every time we call a function, a stack frame is set up and that doesn't require intervention of the operating system. So this could be done at the user level. Okay, so keep in mind that memory protection isolation relies completely on how page tables are managed. Page tables are used to implement address spaces. So management of page tables, whether we update, when a page fault occurs, or switch them, all that has to be done in the operating system. That's the only way we can get the isolation that we're talking about.



Complete Mediation: The TCB

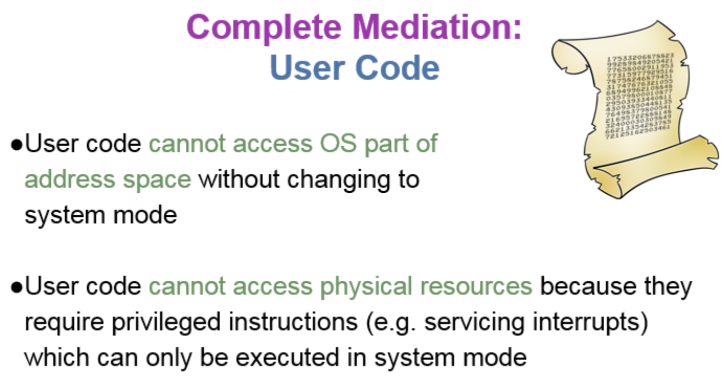
- Make sure that no protected resource (e.g., memory page or file) could be accessed without going through the TCB
- TCB acts as a reference monitor that cannot be bypassed
- Privileged instructions

We had three requirements for a Trusted Computing Base.

Isolation or tamper-proof was one of those requirements. And we've been talking about isolation and memory protection. All that is done to get that tamper-proof requirement. Untrusted code cannot be altering the operating system data and code.

So that was all to do with meeting one of those requirements, which is tamper-proof.

The other requirement is complete mediation that you should not be able to bypass the operating system and go directly to a protected resource. So how do we implement complete mediation? Well, the way we do that is we make sure that no protected resource, whether it's a memory page or a file, could be accessed without going through the Trusted Computing Base or the kernel. Okay, so the TCB is also the kernel. The way we do that is by making the Trusted Computing Base, or having it act as a reference monitor. Any time you have a reference for a particular resource, the TCB has to come into the picture before you can get to the resource, okay. The reference has to be monitored by the Trusted Computing Base. So there should be no way for you to bypass, where one way you can get around complete mediation is by that bypassing the Trusted Computing Base, there should be no way for you to bypass this. So how is that implemented?



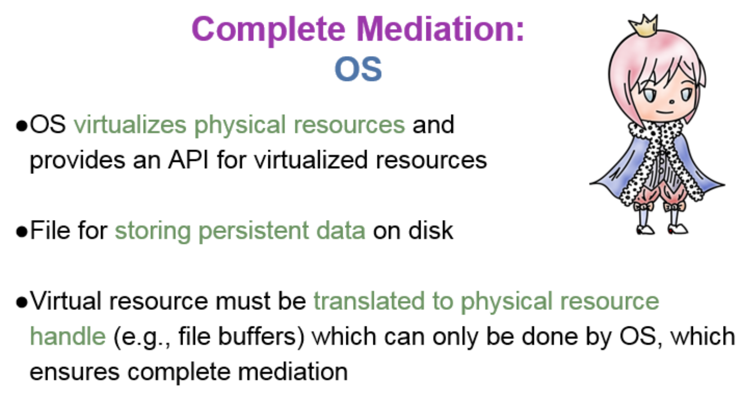
Complete Mediation: User Code

- User code cannot access OS part of address space without changing to system mode
- User code cannot access physical resources because they require privileged instructions (e.g. servicing interrupts) which can only be executed in system mode

Well, we sort of used the same tricks that we've been talking about so far. If you are executing user code, you are in user mode; you're not in system mode. And we know that protected resources are implemented by the operating system.

And so that data structures, the information that is necessary for us to access those protected resources lives in the operating system part of the address base. And we know that while you're executing the user code, you can't access that. You have to switch to the system mode through a system call. And since you can't directly access it, you don't have the information that is necessary for you to access a resource without going through the operating system. Which helps us with this complete mediation property we have.

In addition, the user code can not access physical resources because often times they require execution of privileged instructions. We're talking about service interrupts and things like that. There's no way for you to get to a disk lock. Or change the register that points to a page table, because those are instructions that are privileged. And if you tried executing them in the user mode, your execution would be aborted. You won't be able to continue that. So complete mediation, again relies on the same hardware mechanisms we've been talking about where the isolation mechanism we discussed the idea of privileged instructions that we discussed.



Complete Mediation: OS

- OS virtualizes physical resources and provides an API for virtualized resources
- File for storing persistent data on disk
- Virtual resource must be translated to physical resource handle (e.g., file buffers) which can only be done by OS, which ensures complete mediation

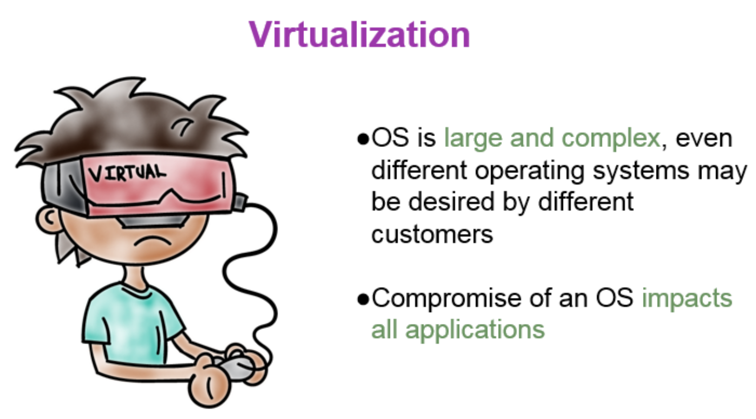
Complete mediation actually comes about, or we have that, because of one other thing the operating system does.

So at user level, we have what we call virtual resources. We don't have physical resources. There is no way for user code to actually name or be able to target a physical resource, or ask for a physical resource. There you have only virtual resources, and the operating system actually gives you an API in how those virtual resources can be used.

An example of this is that for storing persistent data, we have file abstraction, or virtual resource for storing persistent data is a file. So you don't access a disk lock. The operating system does disk IO, and schedule requests to read the blocks, and services interrupts, and things like that. It can interact through the disk controller. Through those instructions we're talking about. But you cannot from user space. You can only ask for access to files either opening them or reading them or writing them. And things like that. So at the user level, we don't have ways of directly talking to the disk controller, the low level hardware functions that operating system has access to. We only have these virtual resources.

As a result of that, remember virtual resources are what the name says. They are not the real resources, they have to be implemented using physical resource. The file has to be implemented with a bunch of disc locks where the file data is stored. So the virtual resource has to be translated to physical handles or resources. For example, buffers or file data is stored on things like that. And this translation is done, from virtual to physical, implementation is done by the operating system. If you want to access a disc block or a phase of memory, you have to start with either a file descriptor, or you start with a logical address we're talking about, or the virtual address. And the translation from virtual to physical is the first step before you can get to the physical. Only the operating system knows how to do that. Or has the metadata, or has the control information, that can be used for this translation. So this level of sort of indirection that we add where you have one kind of namespace and the underlying resources have a different namespace. And the translation can only be done by the operating system, or the Trusted Computing Base.

Well that helps you get complete mediation, because you first need to go have the translation done. Okay, then you have to go to the Trusted Computing Base, so the complete mediation gets implemented right then and there, because you're coming to it, and asking for a physical resource. So it could be the reference monitor. It can check at that point, that whatever that you are asking for, do you have access to it?



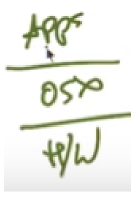
Virtualization

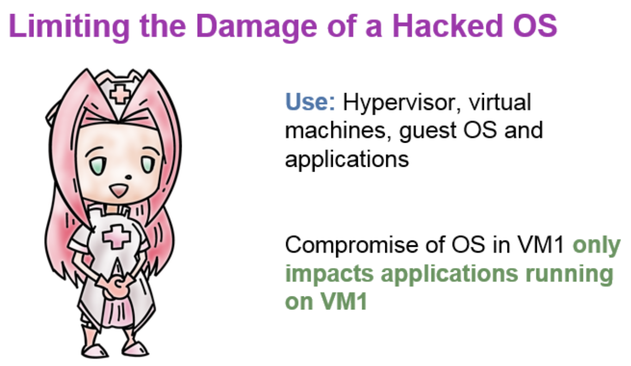
- OS is large and complex, even different operating systems may be desired by different customers
- Compromise of an OS impacts all applications

So, talking about virtual resources, there's a lot of talk about virtualization. We talked about cloud computing, and virtualization and things like that. Once you sort of have these virtual resources and implement them using whatever underlying physical resources you have, so there's no end to what kinds of things you can do. We said we had to go through this translation, and someone, maybe it's, but that can be done at many levels, so I thought we'll talk a little bit about virtualization, and the context of the security and protection is our main interest here. So we're not talking about virtualization for the sake of virtualization, but how does this concept actually relate to isolation and relate to protection of resources and securing them against unauthorized access and things like that?

So, main motivation for virtualization comes from the fact that operating system is large and complex. And the different operating systems, different applications may require different underlying operating systems. So one use of virtualization commercially is actually to not be limited to just one operating system and be able to run multiple operating systems. But as I said before we going to look at value of this idea when it comes to protection and security.

So, remember one other thing when you have a single operating system supporting all of your applications is that if the operating system is compromised, then it's going to impact all applications. Every application that you have running in that system is going to be affected by this compromise of the operating system because they all have this one underlying common operating system. Our picture was, we have all our apps, and then we have our operating system and the hardware. If something goes wrong here, of course, all of this is affected.

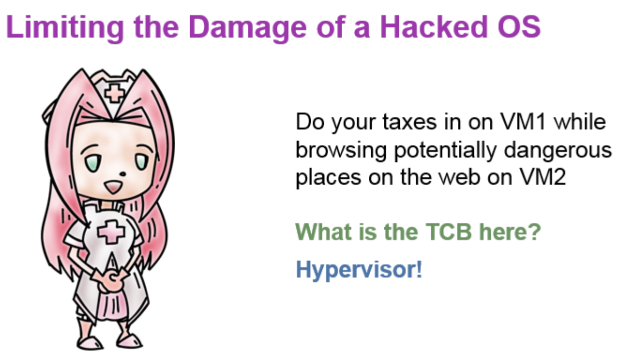




Limiting the Damage of a Hacked OS

Use: Hypervisor, virtual machines, guest OS and applications

Compromise of OS in VM1 only impacts applications running on VM1



Limiting the Damage of a Hacked OS

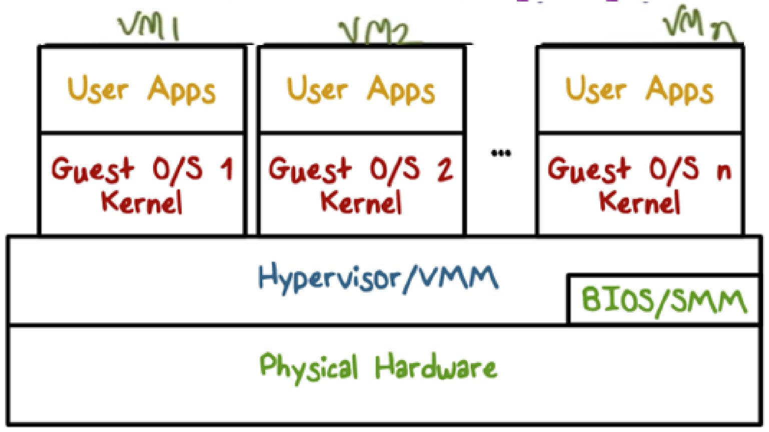
Do your taxes in on VM1 while browsing potentially dangerous places on the web on VM2

What is the TCB here?

Hypervisor!

Virtualization allows isolation between VMs.

Virtualization Security Layers



So here is a good picture of how virtualization is related to security and isolation and things like that. This is your physical hardware. On top of that, we have this hypervisor. It's also called a virtual machine monitor, or VMM. And on top of that, we have this different VM. So, this is VM1. This is VM2. This is VMn let's say if you have one of n of those. If you somehow compromised this operating system [VMn], it only effects the user applications we have here [in VMn]. It does nothing to the user applications we have running in other virtual machines. This is what we're saying, is your Trusted Computing Base [Hypervisor]. As long as you can trust this [Hypervisor], the isolation across virtual machines is going to work. So, compromise of this operating system is not going to affect these other virtual machines and the applications running in those. For us, virtualization is important because our focus is in security and you see this isolation of these applications. Earlier they all shared the same operating system. Now they only share a Hypervisor. Why might that give you better security? We're going to talk about that next.

Correctness: The Final TCB Requirement

- Compromise of OS (TCB) means an **attacker has access to everything.**
- Getting the TCB right is extremely important
- **Smaller and simpler** (hypervisor only partitions physical resources among VMs and let us guest OS handle management)
- **Secure coding** is really important when writing the OS which typically is written in languages that are not type safe

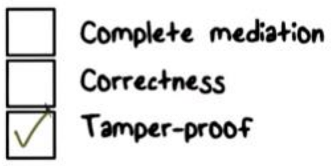
TCB Requirements Quiz

An attack that exploits a vulnerability in an operating system **turns off the check** that is performed before access to a protected resource is granted.

What **TCB requirement is violated** as a result of this attack?

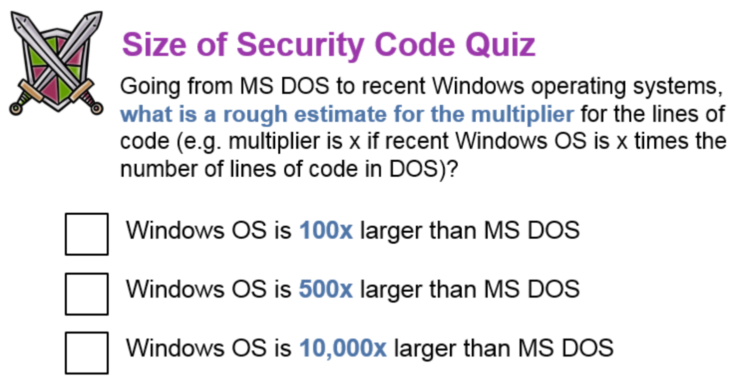
☐ Complete mediation
☐ Correctness
☐ Tamper-proof

So I think we going to finish it off with a quiz. And that's going to bring back sort of all the TCB, trust with the competing race requirements they're talking about. And give us a chance to think about them. So the first one is saying an attack that exploits a vulnerability in an operating system turns off the check. The access control check, that we're talking about. That determines whether the target resource or the resource that's been requested. Access to it should be granted to the source of the request or not If you turn that off, which TCB requirement are you violating?



☐ Complete mediation
☐ Correctness
[✓ Tamper-proof]

Well, you actually go into the operating system, so it's still mediating. You have actually turned off the check. I'm going to say that you tampered with the TCB, so it's the tamper proof requirement has been violated. As a result of that, it's no longer functioning correctly. So correctness is not going to be there, but I think the reason that the violation that we first run into it is, that it's been altered or tampered with, because we have turned off this check. So the answer to this should be the Tamper-proof.



Size of Security Code Quiz

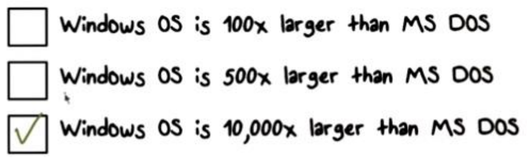
Going from MS DOS to recent Windows operating systems, **what is a rough estimate for the multiplier** for the lines of code (e.g. multiplier is x if recent Windows OS is x times the number of lines of code in DOS)?

☐ Windows OS is **100x** larger than MS DOS

☐ Windows OS is **500x** larger than MS DOS

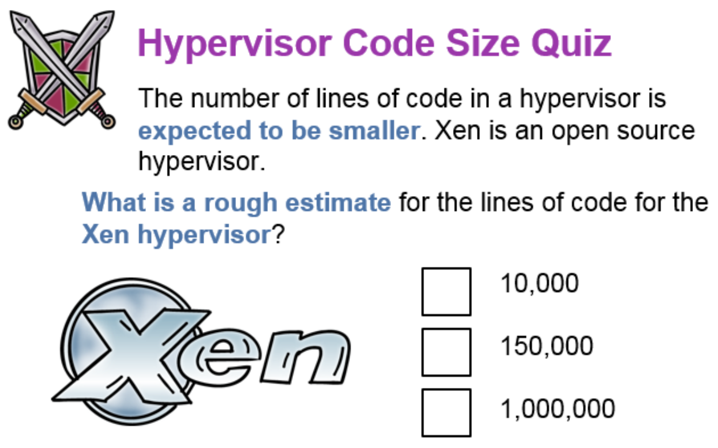
☐ Windows OS is **10,000x** larger than MS DOS

Next question is related to correctness, and we know size and correctness don't go together. Larger the size, harder it is going to be for us to do it correctly. So, we sort of go back in time, saying, start with MS Dos, to the more recent Windows operating system, they offer richer functionality. They are much more complex as a result. So the question is really rough estimate of the multiplier, the lines of codes. I'm going to assume that lines of code in some sense capture the complexity of a system and the larger the system is more complex it is, more lines of code it's going to have. So, multiplier is x if recent Windows OS is x times the number of lines of code and the recent Windows operating system is x times that the lines of code we had in DOS.



☐ Windows OS is 100x larger than MS DOS
☐ Windows OS is 500x larger than MS DOS
[✓ Windows OS is 10,000x larger than MS DOS]

So, how much more complex has this system become? 100 times? 500 times? 10,000 times? I think if I remember right, the first version of DOS was about 5,000 lines of code. I said assembly code, but about 5,000. Windows currently is running in the tens of millions range. If you look at more recent Windows-based operating systems, 50 million or whatever it is, we're talking rough estimates here, so it's tens of millions. So if you go from 5000 to tens of millions, the complexity has grown by a factor of 10,000. So one of the things people say, why do we have these vulnerabilities? Well, it's complexity, and it's complexity because we have all these fancy features, and richer interfaces, and APIs that are more general, and things like that. And that adds to the complexity. So it relates to correctness. Getting operating systems correct is a challenge because they're complex. And here is an example of how the complexity has grown over time.



Hypervisor Code Size Quiz

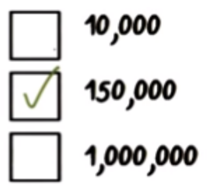
The number of lines of code in a hypervisor is **expected to be smaller**. Xen is an open source hypervisor.

What is a rough estimate for the lines of code for the **Xen hypervisor**?

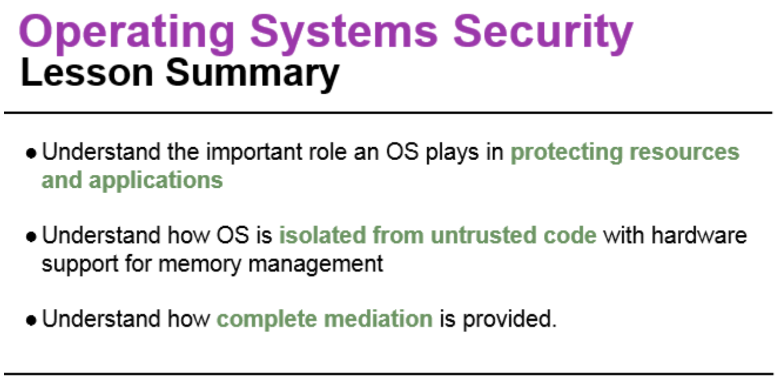
- ☐ 10,000
- ☐ 150,000
- ☐ 1,000,000

So when you have virtualization, of course, the Trusted Computing Base becomes the hypervisor. That comes between the hardware and the virtual machines. So one popular open source hypervisor is Xen. So this one is saying, we know that Windows is millions of lines of code. So it's saying well, in compared to that, if you have a hypervisor. How small is it? What's the rough estimate for the lines of code that the Xen hypervisor has?

Well, 10,000 is kind of too small but it's not quite a million. The answer is 150,000. That's relatively small for something that has to essentially take the hardware resources, do the direct management and make those available to the various virtual machines and so on. So this Xen is about this size, which is relatively small compared to tens of millions that other operating systems. Their size is in those multi-million numbers.



- ☐ 10,000
- [✓ 150,000]
- ☐ 1,000,000



Operating Systems Security
Lesson Summary

- Understand the important role an OS plays in **protecting resources and applications**
- Understand how OS is **isolated from untrusted code** with hardware support for memory management
- Understand how **complete mediation** is provided.

So we saw that the operating system is really the Trusted Computing Base, and it has to be isolated from untrusted code. We saw the hardware features and how they could be used to achieve this. We saw how hardware support sets its process execution modes. And memory protection are used to meet the complete mediation and tamper resistance or tamper-proof property that we require from a Trusted Computing Base.