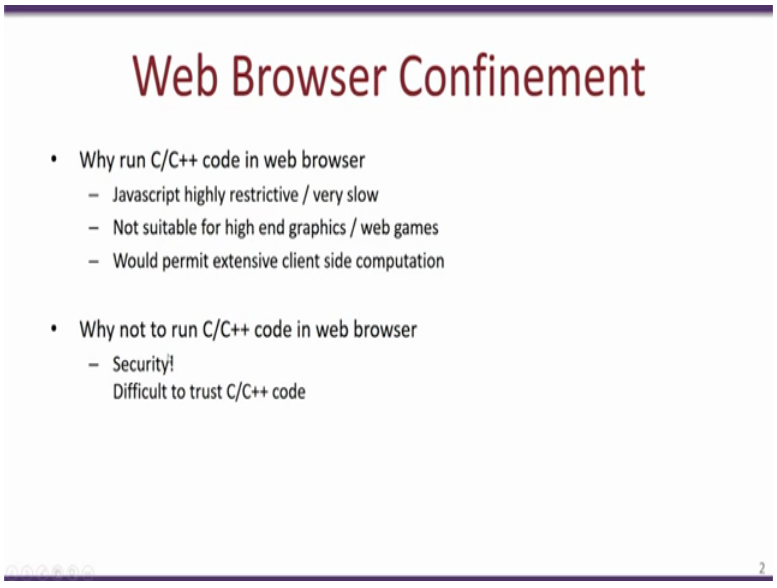
Information Security 5 Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Lecture 32
Software Fault Isolation

Hello and welcome to this lecture in the course for secure systems engineering. In the previous lecture we had looked at one particular problem called confinement and we had seen how a web server which we will which was called OKWS was designed with the principle of least privileges so that the surface with which an attacker in attack the system is drastically reduced, we seen how OKWS used a lot of support that Unix provides with the various access control policies that are present in the Unix operating system to provide an infrastructure where the surface is reduced.

So what we will be looking at is something known as software fault isolation. So before we go into what software fault isolation is we will look at particular use case of a web browser. So note that while the previous lecture looked at the web server we looked at the client aspect of the web server that is a web browser. So all of you I am sure must have used a web browsers and what you would have noticed that programming languages used in web browsers or to actually display contents in web browsers are very much different from the regular C or C++ type of programs that are typically used to write applications.

(Refer Slide Time: 01:56)


Web Browser Confinement

- Why run C/C++ code in web browser
  - Javascript highly restrictive / very slow
  - Not suitable for high end graphics / web games
  - Would permit extensive client side computation
- Why not to run C/C++ code in web browser
  - Security!
    Difficult to trust C/C++ code

2

So we will start with this particular application of a web browser, so as you know a web browser connects to a web server such as web server such as say google.com and it downloads the contents of a web page and essentially displays that contents also there are some programming languages or which would allow you to run programs on your local machine so these programs or for example written in JavaScript will be downloaded from the web server into your machine through the web browser and then executed in your local machine.

One question that actually comes to our mind at this particular point of time is why do we have a special programming language for web browsers? Why do we actually have to have some programming language like JavaScript to be used with web browsers? Why cannot we actually use regular C and C++ type of programs with web browsers? So in such a case what would happen is your web browser will connect to a web server download a program written in say C and C++ that program or that executable would then execute through your web browser in your system.

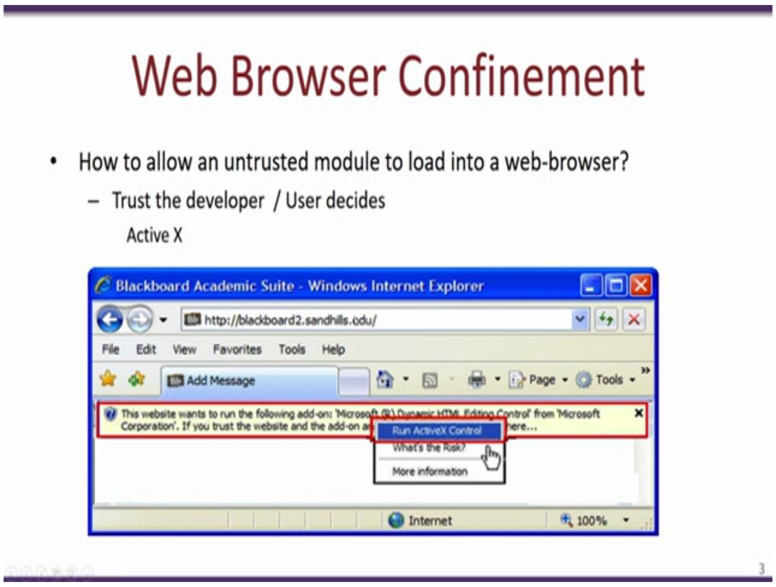
The problem with this particular model is that there is a huge security concern right with C and C++ programs you can achieve a lot of system that will aspects, you would be able to for example manipulate a lot of files, delete files you could see a lot of system processes that is running and so on, you could directly invoke system calls and this could actually lead to a lot of problems.

So the big reason why we do not want to run C and C++ code in a web browser is security, so essentially it is very difficult for us to trust C and C++ code therefore in order to achieve a more secure environment where the programs that you run through your browser is limited to what it can actually do we use programming languages like JavaScript, so JavaScript is designed to be highly restrictive and as many of you who would have programmed with JavaScript you would be aware that the compared to a C and C++ type of program the program the programming API is our highly limited and therefore you would not be able to do a lot of system a level tasks and as a result your client system which is running your web browser is essentially protected.

The huge problem with actually running JavaScript in your web browser is that JavaScript is extremely slow and therefore it is not suited for high end graphic web pages or for example if you are running games over the web, so using JavaScript would be essentially a huge disadvantage because the rendering would be extremely slow, so keeping this in mind one would want to run C and C++ code in your web browser.

So we have a problem here while at one site for some applications like high end graphics C and C++ code is preferred in the web browser so that you achieve the right amount of performance you would be able to render your graphics and player web games very smoothly but on the other hand running C and C++ code in a web browser could result in huge security concerns.

(Refer Slide Time: 06:02)



Web Browser Confinement

- How to allow an untrusted module to load into a web-browser?
  - Trust the developer / User decides

    Active X

Blackboard Academic Suite - Windows Internet Explorer

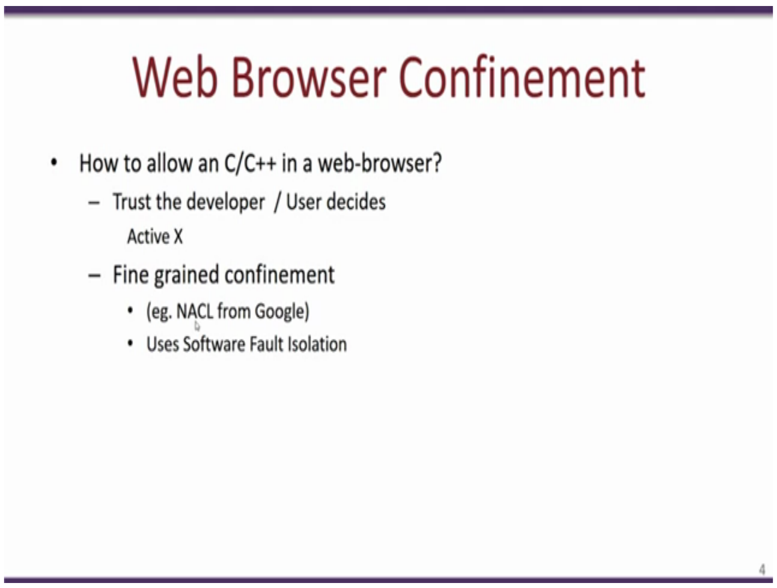
http://blackboard2.sandhills.odu/

This website wants to run the following add-on: 'Microsoft (R) Dynamic HTML Editing Control' from 'Microsoft Corporation'. If you trust the website and the add-on a[illegible]here...

So one way within which this problem was handled in the past was by means of an ActiveX component, so an ActiveX component especially in a Microsoft Windows type of operating systems would permit a developer to develop code in C++ or which VC++ or something like that and the ActiveX component could be invoked from the web browser, this particular figure over here would show how the Internet Explorer would pop up a message stating that this particular website wants to run an ActiveX component.

Now if the user clicks on this and the user says that the ActiveX component can run then what would happen is that the ActiveX component which is written in C and C++ would be downloaded from the web server into this local machine where this Internet Explorer is present and that ActiveX component will execute. Now essentially as we see this is not a very secure way of doing things because the only protection that is obtained or we only security that is obtained is this particular popup message where the browser requests the user to actually take the decision whether the ActiveX components should run or should not run in the system and since many users are unaware of the security aspects or the vulnerabilities or the security issues that are linked with ActiveX components, most users would actually click on this run ActiveX component and this could result in their system being compromised.

(Refer Slide Time: 07:55)



Web Browser Confinement

- How to allow an C/C++ in a web-browser?
  - Trust the developer / User decides

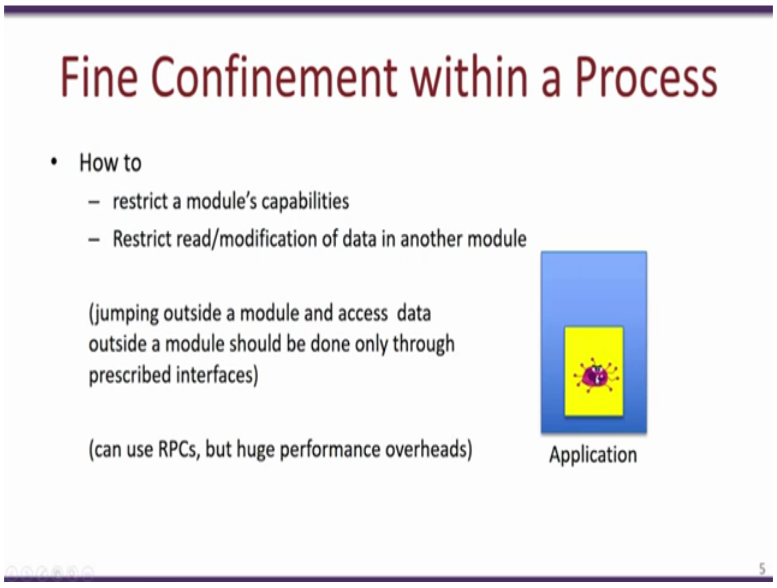
    Active X
  - Fine grained confinement
    - (eg. NACL from Google)
    - Uses Software Fault Isolation

4

What we will be seeing today is a better way or to actually handle this situation, so what will be looking at is something known as Fine grained confinement where the framework or the web browser would provide a particular platform by which C and C++ code can be executed in a web browser in a very protected environment or in a very confined environment, so till a few years back the Google Chrome web browser used some use something known as native client or NACL in their browsers to actually achieve this Fine grained confinement.

So let us see what a Fine grained confinement is, so we will be actually discussing a particular paper known as software fault isolation.

(Refer Slide Time: 08:47)



Fine Confinement within a Process

- How to
  - restrict a module's capabilities
  - Restrict read/modification of data in another module

(jumping outside a module and access data outside a module should be done only through prescribed interfaces)

(can use RPCs, but huge performance overheads)

Application

Now what we achieve with Fine grained confinement is that we have an application over here, now this application you could consider this as a process and as we know within a process any function can invoke any other function, Similarly any function within this particular process can access any global data or data present in the heap of this entire process space.

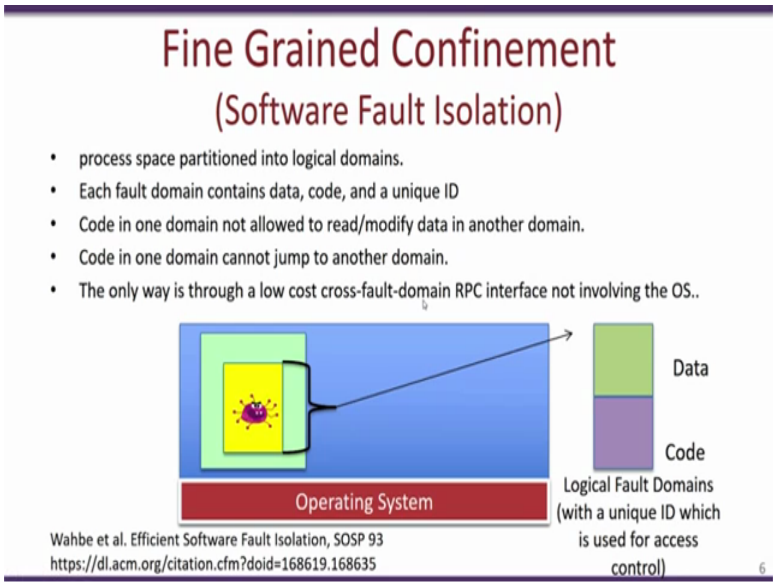
Now with a Fine grained confinement to within a process what we actually achieved is creating one compartment like this, so code that runs within this compartment is restricted by these boundaries of this particular compartment. So the infrastructure that we provide would ensure that when a function that a function executing in this confined environment cannot directly invoke any function outside this environment, similarly a function present inside this particular confined environment would not be able to directly access any global variable or heap data present outside this confined area.

On the other hand functions one function in this confined area could directly call another function in this area or access data in the global space or heap in this particular confined area essentially in order to achieve this confined infrastructure we would require to address two aspects first how would we restrict a modules capabilities? So we call this a particular confined module how would we ensure that a function over here cannot access of another function or data outside this particular module, second how would be restrict reading and writing of data outside this particular module.

So one solution for this as we have seen in the previous lecture what OKWS did was by using remote procedure calls what we would do is we would keep this confined area as a totally different process and then we would use remote procedure calls between these two processes to obtain the confinement however the problem over here is that RPCs have a huge overhead, every time you want to invoke a remote procedure call from one process to another process there is a context switch required and therefore the operating system gets invoked the OS would ensure that the RPC is transferred to the right process the process would execute it is required function and then there is another context switch from this particular process back to the callee process.

So every RPC involves two context switches one to send the request for the remote procedure call and the other one to actually obtain the return values, each context switch is very heavy and therefore would result in performance overheads. So now with this fine grained confinement which will actually discuss in this particular lecture what we would be able to do is obtain one address space and within this address space so we would create a closed compartment where code and data executing in this confined area is restricted by the walls of this compartment.

(Refer Slide Time: 12:37)



Fine Grained Confinement

(Software Fault Isolation)

- process space partitioned into logical domains.
- Each fault domain contains data, code, and a unique ID
- Code in one domain not allowed to read/modify data in another domain.
- Code in one domain cannot jump to another domain.
- The only way is through a low cost cross-fault-domain RPC interface not involving the OS..

Wahbe et al. Efficient Software Fault Isolation, SOSP 93
https://dl.acm.org/citation.cfm?doid=168619.168635

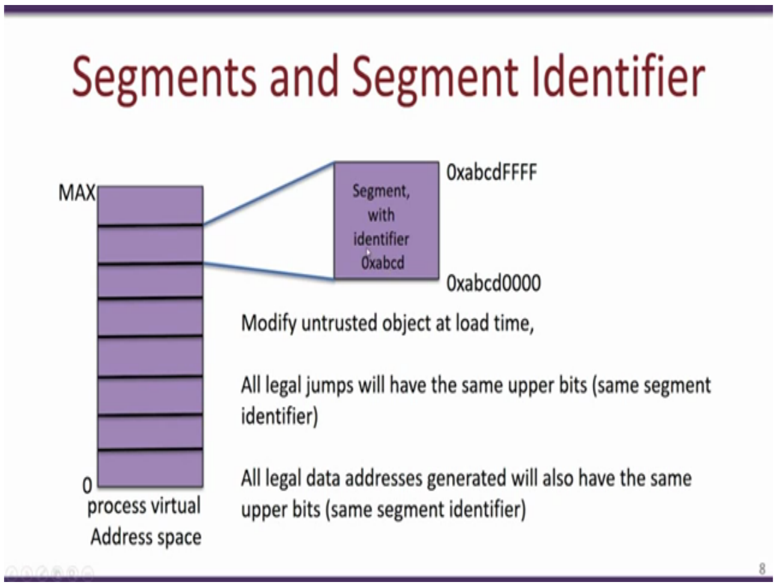
Data
Code
Logical Fault Domains (with a unique ID which is used for access control)
Operating System
6

The techniques that we will be actually discussing in this particular lecture is present in this paper efficient software fault isolation in SOSP in 1993. So what we do here is the following first the process space that is this part is partitioned in two various logical domains, each logical domain is known as a fault domain and each of these domains comprises of data and code as shown over here, so we have one fault domain over here and this particular fault domain comprises of data and code further each fault domain is given a unique ID.

Now what the software fault isolation framework achieves is that code that is executing within this fault domain is restricted to only this code area and this particular data and any jumps or any function invocation outside this particular fall domain would be caught or prevented. Now the only way by which a function can invoke another function outside the fault domain is through something known as a low cost cross fault domain RPCs.

So unlike the regular RPCs which we discussed previously this low cost fall domain RPC does not involve any context switch rather the OS is not involved at all and therefore the overheads incurred by this particular fall domain is extremely less.

(Refer Slide Time: 14:22)



Segments and Segment Identifier

MAX
0
process virtual Address space
Segment, with identifier 0xabcd
0xabcdFFFF
0xabcd0000

Modify untrusted object at load time,

All legal jumps will have the same upper bits (same segment identifier)

All legal data addresses generated will also have the same upper bits (same segment identifier)

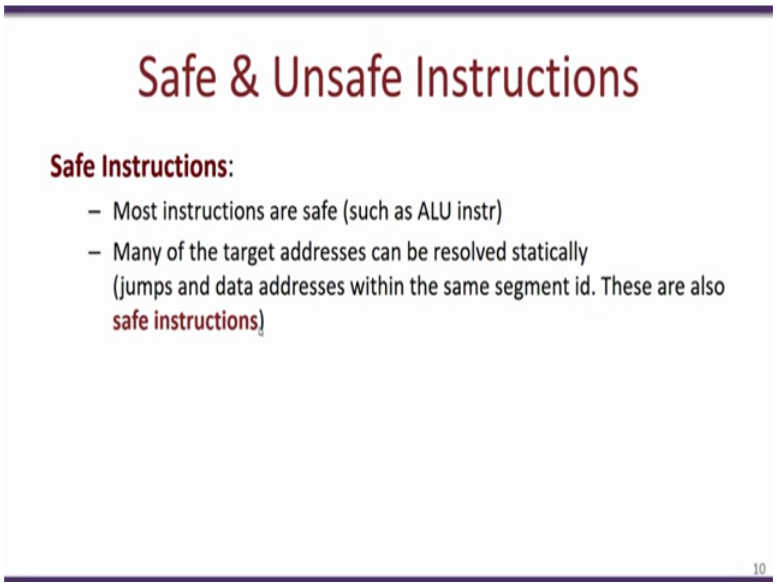
Now as we know a process comprises of a virtual address space which starts at a 0th location and then extends to a maximum location, so this is the user space of the process. Now we would divide this particular user space into several different segments so what we do is align the segments in such a way that the higher order bits within each memory location within this segment are the same for example we define a segment 0xabcd so this 0Xabcd is the segment identifier or this is the unique identifier for that particular segment.

So this segment starts at the location 0Xabcd0000 and extends up to 0XabcdFFFF, so what we would see is that every memory location within the segment starts with 0Xabcd. So what the software fault isolation framework achieves is that any branch or jump instruction within the segment is to a location in the same segment so this is done by ensuring or checking that the higher order bits of the target address is 0Xabcd similarly every data access by an instruction in this particular segment can be done to a memory location which has an address starting with 0Xabcd.

Now if we are able to achieve this and restrict every branch call or a jump instruction as well as restrict every memory access to locations which start with 0Xabcd then we will be able to confine the instructions within this particular segment to the locations starting from 0Xabcd0000 and extending up to 64 kilobytes from this address. So how do we actually ensure that such jumps or memory accesses are not permitted within a segment.

So there are a couple of ways to do this, so what we do is that we modify the untrusted executable or object at the load time so what we do is we while loading a particular object into this segment we parse that particular object look out for all the jump instructions or all the branches all the call instructions and ensure that the target address for those particular branch instructions all are within this particular segment, so therefore we call this as legal jumps.

(Refer Slide Time: 17:28)



Safe & Unsafe Instructions

Safe Instructions:

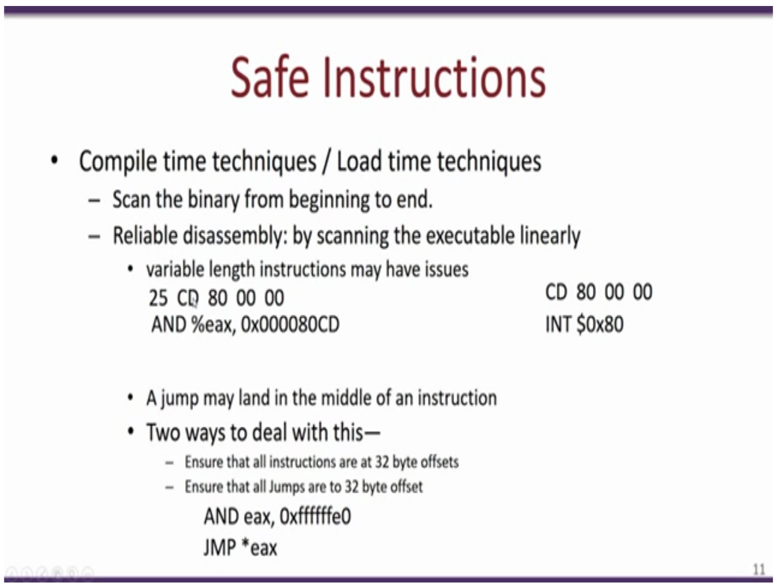
- Most instructions are safe (such as ALU instr)
- Many of the target addresses can be resolved statically (jumps and data addresses within the same segment id. These are also safe instructions)

10

So what we do is at the time of loading pass through the entire object file which is going to be loaded in a particular segment so and identify that every branch instruction is to a legal address, every load and store instruction is also to a legal address. So what we do is we define two types of instructions they are the safe instructions and the unsafe instructions, so most of the instructions are safe instructions.

So instructions such as the ALU instructions or comparison instructions, floating-point instructions and so on are considered as safe instructions further branch or call instructions whose target addresses can be resolved statically that is at load time or at compile time they are also called safe instructions similarly every load and store instruction whose memory address the memory address which they want to load or store can be resolved at the time of loading or compiling so these are also called as safe instructions.

(Refer Slide Time: 18:43)



Safe Instructions

- Compile time techniques / Load time techniques
  - Scan the binary from beginning to end.
  - Reliable disassembly: by scanning the executable linearly
    - variable length instructions may have issues

```
25 CD 80 00 00                  CD 80 00 00
AND %eax, 0x000080CD            INT $0x80
```

    - A jump may land in the middle of an instruction
    - Two ways to deal with this—
      - Ensure that all instructions are at 32 byte offsets
      - Ensure that all Jumps are to 32 byte offset

```
AND eax, 0xffffffe0
JMP *eax
```

11

So what we actually do is that at the time of compiling or during the time of loading when the object is loaded into a particular segment so what we do is that we open the binary file and start to scan that binary file from the beginning to the end, so what we do is we take the binary code disassemble it and get the corresponding instructions. Now we ensure that the instructions are safe instructions.

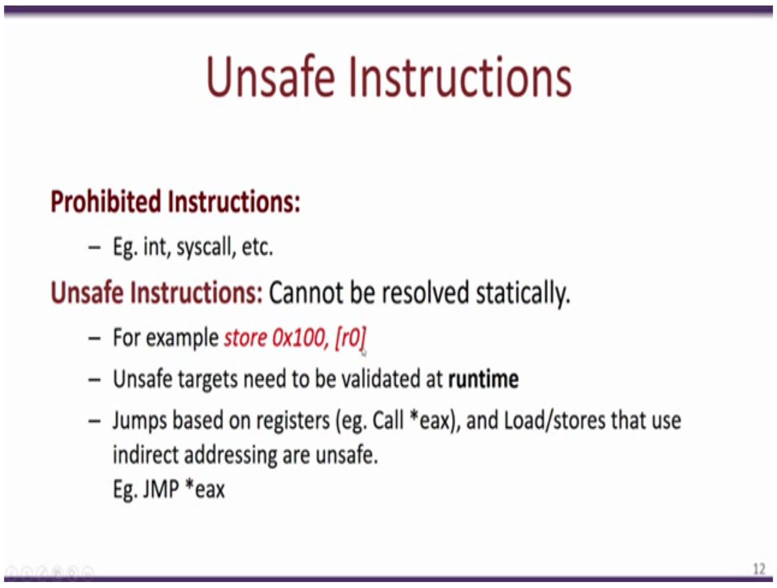
So if all instructions are safe instructions then we can permit the object file to be loaded from that particular segment however we need to also take care of the fact that instructions can be descended assembled in multiple different ways for example if we have a sequence of binary data like this 25 CD 80 followed by 00 00 this can get disassembled to AND eax comma 0x000080CD however if we start the disassembly from this location as in here CD 80 00 00 then we have an interrupt instruction and this is an interrupt for a system call.

Now what we see is that this AND instruction is safe but however these interrupt instruction is unsafe therefore while doing the disassembly we need to ensure that such unsafe instructions are not present. The problem that could occur over here is that somewhere in the particular object file which we are loading into that segment there could be a jump instruction to this memory address corresponding to this CD such a thing happens then we obtain an interrupt which is going to be unsafe.

So we need to ensure that every branch instruction not only branches to some target address wins inside that particular segment but also branches to a target address where a legal instruction is present, a legal instruction such as this and not this. Now in order to deal with this second aspect what we do is that we ensure that all instructions start at 32 byte offsets, so therefore if we have a sequence of such binary data corresponding to a correct instruction like the AND instruction present here we would ensure that the 25 over here starts at an offset of 32 bytes thus any jump from the segment can be only done to a 32 byte offset.

So this can be easily done or while scanning the particular binary every time we see a jump instruction we should need to ensure that the higher order bits of that jump instruction have the correct segment value secondly we need to also ensure that the lower 5 bits are set to 0 for the target address so this will ensure that the destination target address always contains a legal instruction.

(Refer Slide Time: 22:09)



Unsafe Instructions

Prohibited Instructions:
- Eg. int, syscall, etc.

Unsafe Instructions: Cannot be resolved statically.
- For example *store 0x100, [r0]*
- Unsafe targets need to be validated at **runtime**
- Jumps based on registers (eg. Call *eax), and Load/stores that use indirect addressing are unsafe.
  Eg. JMP *eax

12

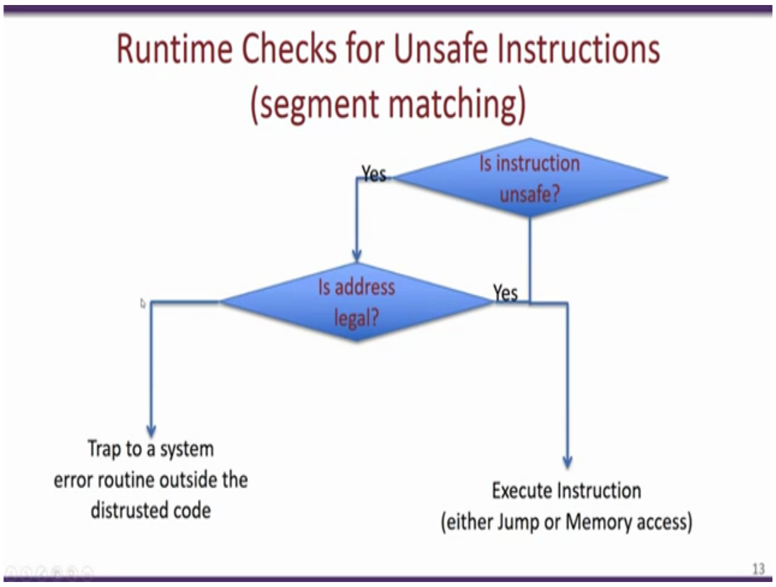
Besides the safe instructions there are certain instructions which are prohibited any instructions like an int which stands for an interrupt or assist call which stands for a system call is prohibited. So while scanning the binary file we need to look out for instructions which are interrupts or system calls so which could transfer execution outside that particular compartment and into the operating system, so these kinds of instructions are not present.

Another form of instructions are known as unsafe instructions, so these instructions are called instructions or branch instructions so besides the prohibited instructions while scanning the binary of the object file we may also come across unsafe instructions so these unsafe instructions are instructions whose target address cannot be resolved statically and they can only be resolved at runtime a nice example of an unsafe instruction is this indirect store instruction where the value of 0x100 is stored in the memory pointed to by r nought, so r naught has some address stored in it and in that corresponding address the value of 0x100 is stored.

Now the problem here and why it is unsafe is that statically we may not be able to determine what the contents of R naught is and therefore we need to wait till runtime to identify or resolve the target address for this store instructions therefore this forms an unsafe instructions. Similarly we could have indirect branches such as jump percentage eax over here which is also an unsafe instruction.

So in this particular instruction the program would branch to a memory location depending on the contents of the eax register, the target address for this particular instruction can be only resolved at runtime and therefore this instruction is also an unsafe instruction.

(Refer Slide Time: 24:30)



Runtime Checks for Unsafe Instructions (segment matching)

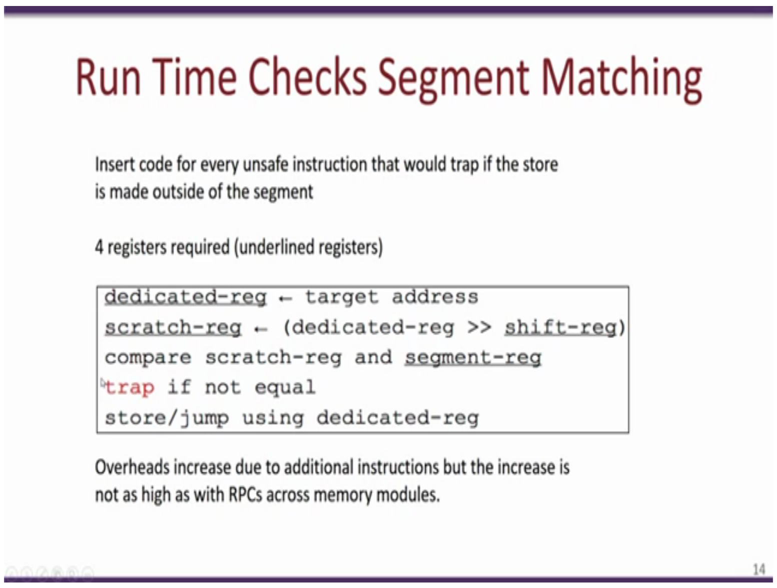
Is instruction unsafe?
Yes
Is address legal?
Yes
Trap to a system error routine outside the distrusted code
Execute Instruction (either Jump or Memory access)
13

The way to resolve the unsafe instructions is by doing runtime checks. So there are two ways in which we could do runtime check one is known as segment matching and the other one is known as a dressed sandboxing. The scheme for segment matching is as follows so what we do is every time we scan the binary file for the object we ensure that every instruction is safe secondly none of the instructions are prohibited third if the instruction is unsafe then we add some code into that particular object file or we add some certain instructions into that object file, so as to ensure that there is some runtime checks.

Now this is an example for the segment matching runtime check, so what we do with in this particular scheme is that whenever we find an unsafe load or store or a branch instruction we add some instructions to do this check first we find out whether the instruction is unsafe if it is indeed unsafe then we do a check to identify (wheth) whether the address is indeed a legal address, that is if the target address for that branch or the memory axis the load or store is in fact a legal address.

So a legal address as you may recall would have the same segment ID that is the upper bits of that target address would have that unique segment ID. Now if this check holds true then we permit the execution of that branch or memory axis, on the other hand if this particular check fails then there is a trap and a trap handler is executed typically what would happen is that the object file would terminate.

(Refer Slide Time: 26:34)



Run Time Checks Segment Matching

Insert code for every unsafe instruction that would trap if the store is made outside of the segment

4 registers required (underlined registers)

```
dedicated-reg ← target address
scratch-reg ← (dedicated-reg >> shift-reg)
compare scratch-reg and segment-reg
trap if not equal
store/jump using dedicated-reg
```

Overheads increase due to additional instructions but the increase is not as high as with RPCs across memory modules.

14

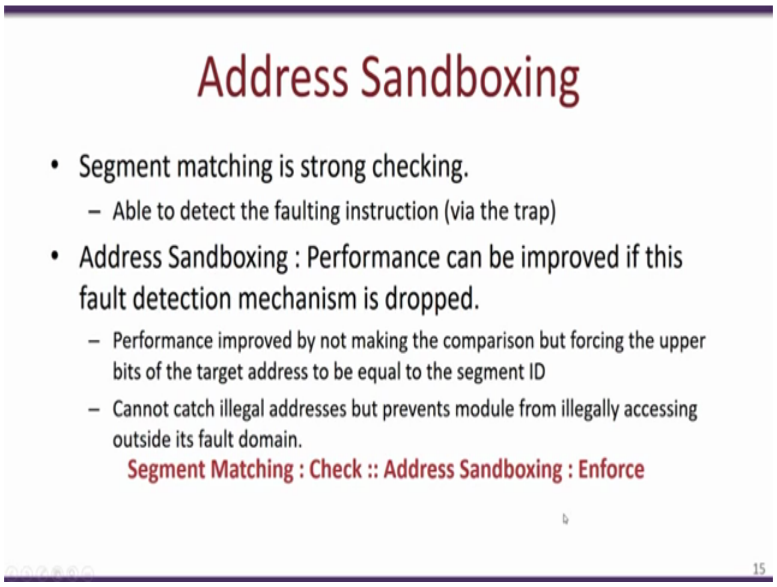
Now going a bit more into detail we would see how we actually do this runtime checks using segment matching. So every time we have a unsafe branch or an unsafe store instruction like this what we do is we take the target address which can be resolved only at runtime and then we load this target address mostly this would be in a register so we would load this target address into some dedicated register, now this dedicated registered is some register in the processor which is typically not used by the compiler then what we do is we provide the check we ensure that the target address is indeed present in the same segment.

The way we do this is we take the (dock) target register shifted by a certain number of bits to ensure that we actually obtain only the unique ID corresponding to the higher bits of that segment and then we store this higher bits in a scratch register, now we compare this value with the segment register which contains the unique segment ID and if they are not equal we trap however if they are equal then we are guaranteed that the target address is indeed a legal target address within the same segment and then we would permit the jump or the store instruction to be performed.

So note that these extra operations or these extra instructions are done every time we see an unsafe store or a branch instruction present in the object that we want to load. So now we note that there could be certain overheads involved over here so we first see that there is a move instruction for the target address to the dedicated register there is a shift instruction required and there is a compare instruction that is required to be added or to be inserted into the object code.

Now these could cause overheads in the program now one thing what you would observe were here is when the trap handler executes it would be able to identify the precise load or store instruction that has violated the segment check.

(Refer Slide Time: 28:58)



Address Sandboxing

- Segment matching is strong checking.
  - Able to detect the faulting instruction (via the trap)
- Address Sandboxing : Performance can be improved if this fault detection mechanism is dropped.
  - Performance improved by not making the comparison but forcing the upper bits of the target address to be equal to the segment ID
  - Cannot catch illegal addresses but prevents module from illegally accessing outside its fault domain.

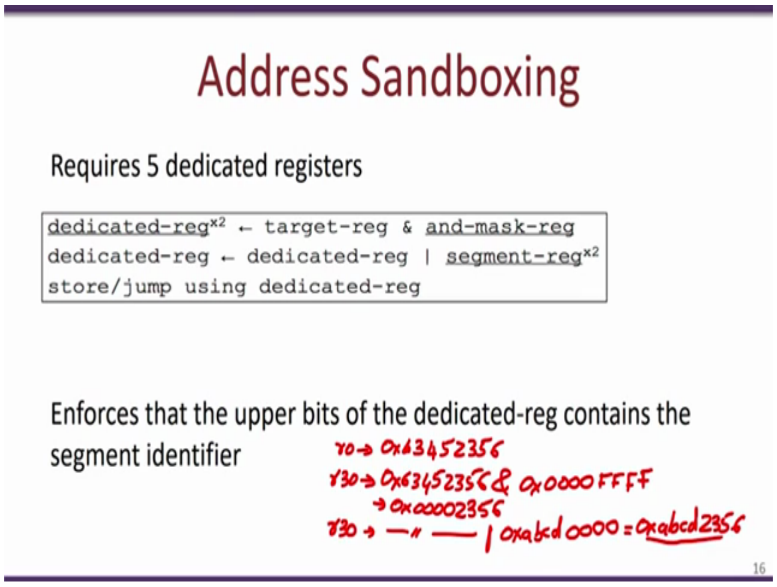
Segment Matching : Check :: Address Sandboxing : Enforce

15

In other words the segment matching is a strong checking, it is not only able to prevent execution or memory accesses outside the closed compartment that is defined but it is also able to identify the instruction which is causing the fault, so this is done via the trap. The second technique is known as the address sandboxing where we will be able to reduce the overheads compared to segment matching but the compromise is that we will not be able to identify the faulty instruction.

So what we were able to achieve in (seg) address sandboxing is that we get a performance improvement so this performance is obtained by enforcing that every branch or memory access is restricted to that segment, so this is very much unlike the segment matching which checks and traps. The address sandboxing on the other hand enforces every branch and let us sees how this is done.

(Refer Slide Time: 30:00)



Address Sandboxing

Requires 5 dedicated registers

```
dedicated-reg^x2 ← target-reg & and-mask-reg
dedicated-reg ← dedicated-reg | segment-reg^x2
store/jump using dedicated-reg
```

Enforces that the upper bits of the dedicated-reg contains the segment identifier

r0 → 0x63452356
r30 → 0x63452356 & 0x0000FFFF
→ 0x00002356
r30 → —"— | 0xabcd0000 = 0xabcd2356

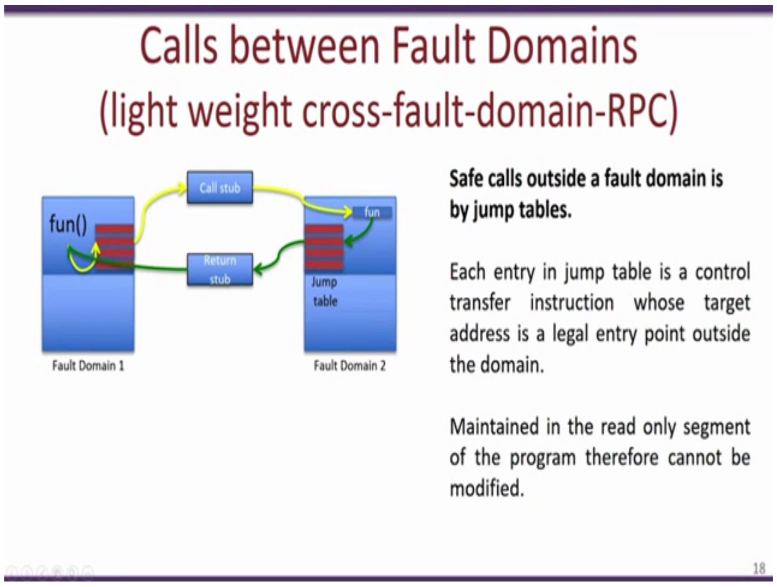
16

Now just like with segment matching we scan the binary of the object and look out for unsafe instructions. Now every time we find an unsafe instructions we insert some code into the binary of that particular object to ensure the corresponding confinement is obtained however unlike the segment matching where we check that the higher bits are indeed equal to the segment ID (he) with address sandboxing we ensure with address sandboxing we set the higher bits of the target address of the unsafe instruction to the segment ID, so this is done as follows.

So what we do is we take the target address which is present in a particular register and we mask it, so let us see how this is done. Let us say a target address present in r nought is some value say 0xb3452356 so what we do is first we apply and mask to it and store the result in a dedicated register such as say r30 or so and we mask this with the AND mask register which looks something like this 36452356 and we end this with 0x0000FFFF, so this would give you something like 0x00002356.

Now in the second instruction we then or it with the segment register so what we achieve is r30 equal to this particular value and or it with 0xabcd0000 so this would be 0xabcd2356 and then we simply store or jump to that particular to this particular location. Now what we see over here is that we are enforcing that every unsafe store or jump is within this particular segment, so note that we are modifying what is done by this unsafe store and jump instruction we are modifying the functionality of that particular object, so however we are able to achieve that any unsafe instruction is always restricted by that segment boundary.

(Refer Slide Time: 33:03)



Calls between Fault Domains (light weight cross-fault-domain-RPC)

Call stub
fun()
fun
Return stub
Jump table
Fault Domain 1
Fault Domain 2

Safe calls outside a fault domain is by jump tables.

Each entry in jump table is a control transfer instruction whose target address is a legal entry point outside the domain.

Maintained in the read only segment of the program therefore cannot be modified.

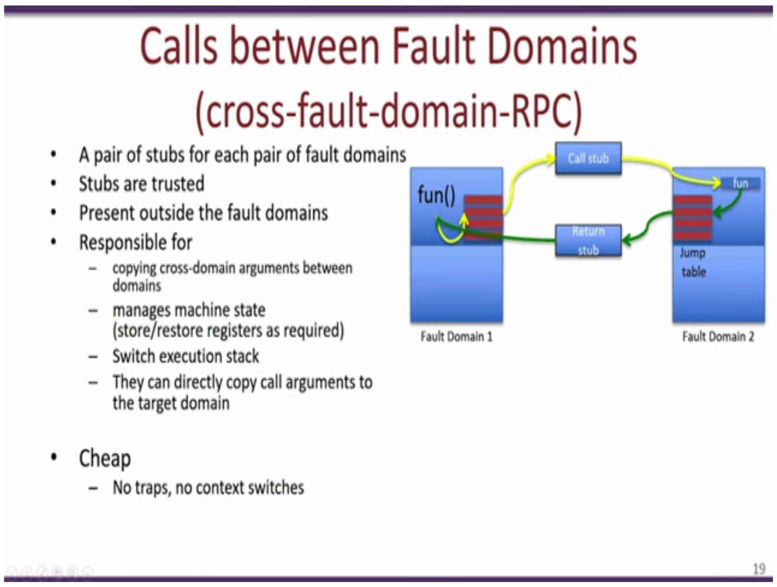
18

So next we will look at the cross fault domain RPCs. So let us say that we have two fault domains, fault domain 1 which is restricted to these locations and fault domain 2 which is restricted to these locations and as we know each of these fault domains would have different segment IDs, now there would be many times where we would want one function present in one fault domain to invoke another function present in another fault domain.

Now we should be able to ensure or that these function invocations are done in a proper and controlled manner now what we do in order to achieve this what we add is something known as a jump table which is present here a called stub and a return stub, so for every legal call there is a lookup in the jump address and then the call stub is invoked, now from the call steb and this jump table we would identify the actual location for the destination function that function would get invoked and through a different path the return is passed back to the present function.

So note that we would now have a proper channel using the stub and return steb to invoke functions outside of all domain.

(Refer Slide Time: 34:30)



Calls between Fault Domains (cross-fault-domain-RPC)

- A pair of stubs for each pair of fault domains
- Stubs are trusted
- Present outside the fault domains
- Responsible for
  - copying cross-domain arguments between domains
  - manages machine state (store/restore registers as required)
  - Switch execution stack
  - They can directly copy call arguments to the target domain
- Cheap
  - No traps, no context switches

Now the assumptions are that these stubs the call stub and the return stubs are trusted, so they would be extremely small pieces of code built by the developer itself and have no bugs or any vulnerabilities which could be utilized by a misbehaving fault domain. Second this stub and return stub are present outside the fault domain so therefore it would not be possible for any function to actually modify the contents of the call stub or the return stub.

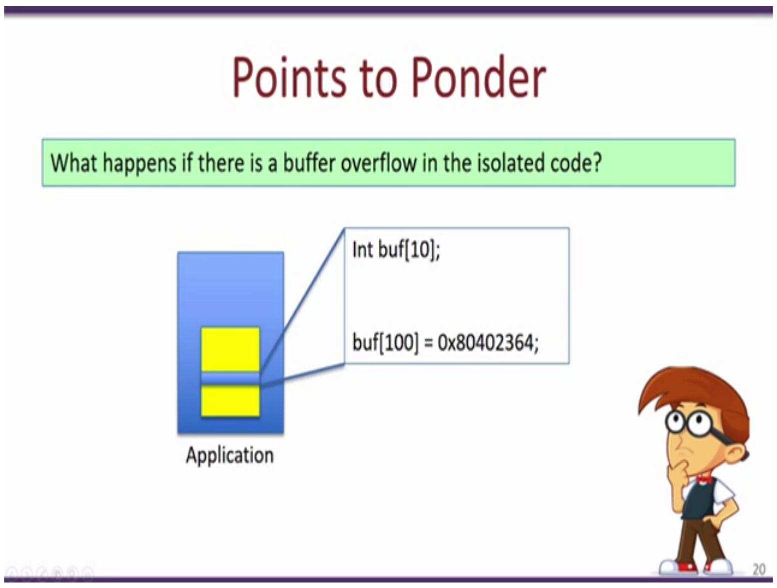
So what actually happens in these stubs is that every time it is invoked the machine state comprising of the registers and so on is stored in a memory location present in this stub similarly the there is a switch of the execution stack and there is a copy of arguments from this fall domain to the other fault domain. so now there are certain properties for these stubs first the stubs are trusted the code comprising of the call stub and the return stub are extremely small and they are well tested and there are no bugs or anyone vulnerabilities present in them, second these stubs are present outside the fall domain.

So by doing this one would ensure that any function present in the fault domain cannot actually modify the call stub and the return stub, so what actually happens in these stubs is that every time there is an invocation the call stuff would store the state of the machine comprising of the registers and so on which present in this particular fall domain and it would also switch the execution stack from this domain to this domain and copy the arguments for that particular function to the other domain.

Now the reverse process occurs during the return, in the return stub the state of all domain 1 is restored and also the execution stack for fall domain 1 is restored, so you see that every time there is a cross domain function call invoked the call stub and the return stub would ensure that there would there is a proper transition from fault domain 1 to fall domain 2 and vice versa.

Now this you would recollect is very much similar to what the operating system does during RPC, so however here unlike the operating system the overheads are very minimal so there are no contexts switch, there are no interrupts that are occurring and so on, so therefore these stub mechanisms (press) present with the software fault isolation is highly efficient compared to the context switches present in the operating system.

(Refer Slide Time: 37:29)



Points to Ponder

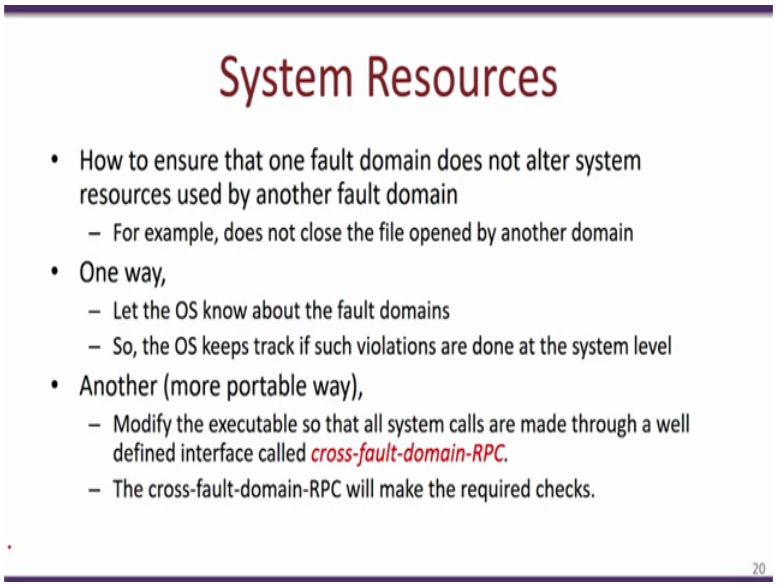
What happens if there is a buffer overflow in the isolated code?

```
Int buf[10];

buf[100] = 0x80402364;
```

So this is something to think about let us assume that we have this application, this is the process application process and within this process we have created this isolated environment and in this isolated environment there are these two statements interrupt buf 10 and buf 100 equal to some particular thing, what would happen in such a case?

(Refer Slide Time: 37:52)



System Resources

- How to ensure that one fault domain does not alter system resources used by another fault domain
  - For example, does not close the file opened by another domain
- One way,
  - Let the OS know about the fault domains
  - So, the OS keeps track if such violations are done at the system level
- Another (more portable way),
  - Modify the executable so that all system calls are made through a well defined interface called *cross-fault-domain-RPC*.
  - The cross-fault-domain-RPC will make the required checks.

20

So another thing to consider is with respect to the system resources how would we ensure that files or other system components which are opened or accessed by one fault domain is not accessed or is protected from another fault domain. So let us say that one fault domain has created a file which is stored on the hard disk, now the operating system a typical Unix like operating system would give permissions based on that particular process, so the file for example will obtain permissions based on that particular process that is executing, the operating system typically is actually unaware of such fault domains that are present in such a scheme.

Now what could happen in such a case is that the second fault domain would also be able to have access to that particular file, so it could for example delete that file or modify that the contents of that file now this is not what is wanted, so whenever we have two fault domains we need to ensure that files or any other system component that is created by one fault domain is not accessible by the code present in the second fault domain even though all of them are in the same process.

So there are two ways to actually handle this situation, so one way is to patch the operating system so that it the OS not only knows about the process but also knows about the various fault domains present in the specific process thus whenever the operating system sees a request for opening a file it would know whether it is coming from fault domain 1 or fault domain two and be able to check access permissions based on this.

Now this would require considerable of modifications in the operating system and therefore also make the particular technique non portable. So there is another thing to a worry about in this entire software fault isolation and that is with respect to system resources. So let us say that we have a one process and these processes have has 2 fault domains, fault domain 1 and fault domain 2.

Now fault domain one creates a particular file so it does thus it does this by invoking a system call let us assume that system calls are present and then the operating system creates the file and stores that particular file on the hard disk. Now we need to ensure that fault domain 2 does not have access to that particular file which has been created by fault domain 1.

Now in a typical scenario this is not possible because the operating system does not know about the various fault domains present within the process it only knows of that particular process, so it would create the file with permissions corresponding to that particular process so in a typical scenario therefore both the fault domain 1 as well as fault domain 2 would have access to that a particular file and this is a problem because fault domain 2 which is maybe untrusted would possibly modify that particular file or delete that file and so on and therefore we need to have a mechanism where such a thing is not present.

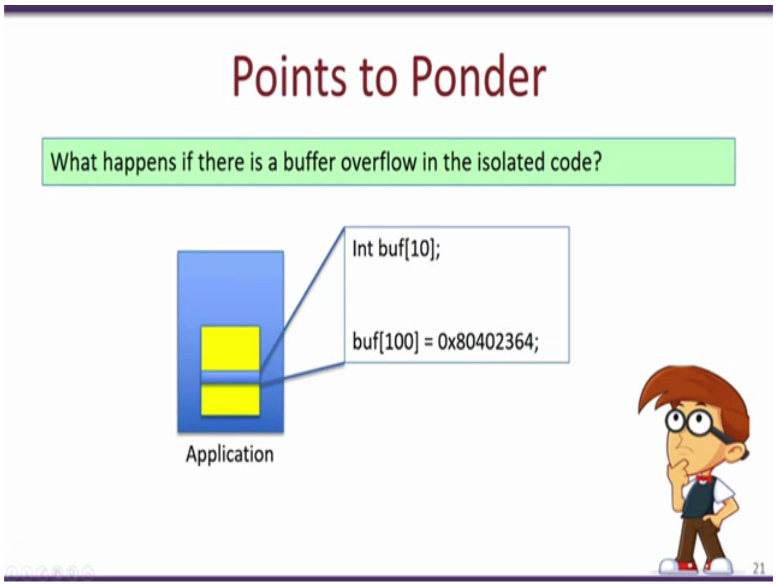
So one trivial way what we have seen before is to prevent any such system calls or interrupts so on and therefore fault domains would never be able to invoke any system call or create any such files and other way as we seen over here is to end to let the operating system know that the process has multiple fault domains thus the operating system has to be modified where files or any other system resources are created in such a way that it gets linked to a particular process as well as the fault domain within that particular process, if the OS is thus aware of all the fault domains present within the process the fault domain 1 would be able to create a particular file which is not accessible by fault domain 2.

However there are two problems with this first it makes the entire scheme a non-portable and secondly it would require consumable rewriting of the operating system a better scheme and a more portable scheme is where we have a separate entity which handles all system calls, so whenever while parsing through the object file one would see a system call or an interrupt, so this interrupt or system call is replaced by a cross fault domain RPC to a particular fault domain which is trusted and handle system calls.

So this particular fault domain would ensure and check whether all system calls are valid so for example if fault domain one wants to create a file it would first result in a cross fault domain to this trusted a fault domain which makes various checks ensures that fault domain has the capability of creating a file and then invokes the system call that would result in the operating system creating a value.

Now if let us say a fault domain 2 request the same file to be opened or modified or deleted this would also result in the cross fault domain RPC our trusted RPC which handles all system calls who then identify that this is not permitted because we have fault domain 2 trying to open a file created by fault domain 1 and therefore it would prevent such a request.

(Refer Slide Time: 43:58)



Points to Ponder

What happens if there is a buffer overflow in the isolated code?

```
Int buf[10];

buf[100] = 0x80402364;
```

Application
21

Now before we conclude, so this is something to think about so let us say we have this one process and within this particular process we have created a confinement area and some function within this confined area has instructions of this form, so it defines a buffer of size 10 and that buf of 100 is having some value, so you need to think about what happens if there is a buffer overflow in the isolated code as shown over here.

So with this we actually conclude the lectures on confinement, we see two ways to actually achieve confinement, one is the OKWS module where we split a large application into several small processes and each process by the virtue of the mechanisms provided by the operating system will be protected from each other. The second way is the software fault isolation mechanism which is far more lightweight and suitable for things like the web browser where we have one application or one process and we are able to create fault domains within that particular process which are secluded compartments within that process.

So these fault domains are restricted by the boundaries provided by that particular fault domain, so in the next lecture we look at another scheme which is known as trusted execution environment, thank you.