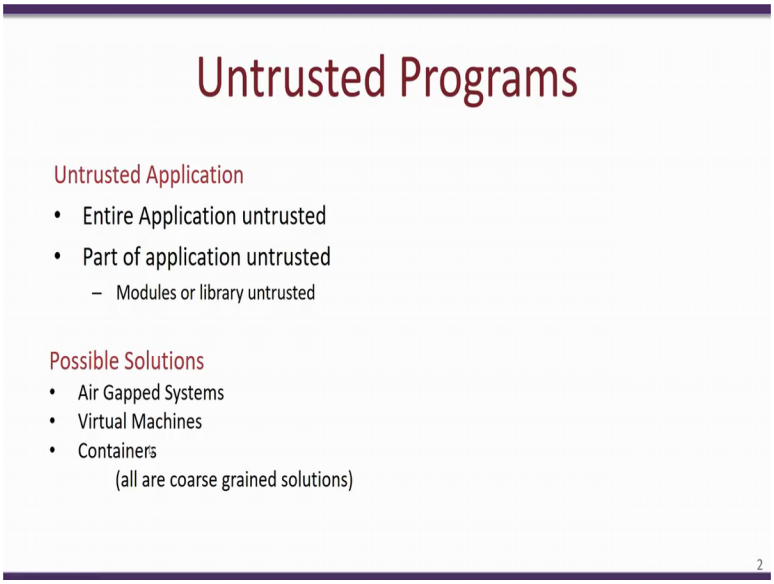
Information Security 5 Secure System Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Confinement in Applications

Hello and welcome to this lecture in the course for secure systems engineering, now let us say that you have downloaded a program from the internet, you are not very certain whether you can trust that program, you are not certain that if you run that program your system may got compromise or let us say you are not certain whether that program may steal secret information or may crash your system, may delete your files and so on.

So these programs which we call as untrusted programs in this particular lecture and the lectures that follow we will see how you could run untrusted programs in your system.

(Refer Slide Time: 01:00)



While this lecture looks at the entire application that is untrusted and how you would run an application which you do not trust in your system while a future lecture would look at modules and libraries which are untrusted, so for example let us that you are running an application and you use a third party module or a third party library which is needed for that application to execute.

Now you may download that module or library from the internet and you are not certain whether you can actually trust that library you are not certain for example if that is a malicious code

present in that library, so the question comes is how would you run your application in spite of not trusting the modules and the libraries that the applications uses, so there are some obvious solutions for this thing.

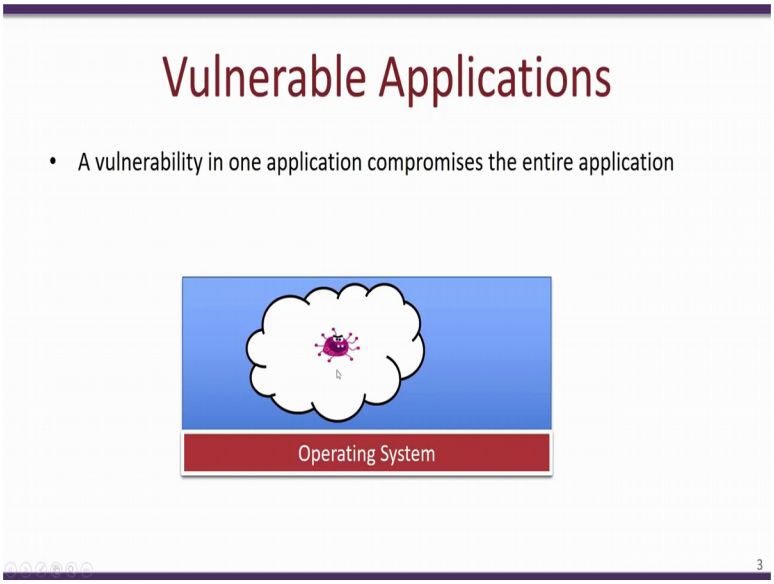
So the first is to use air gapped systems, so what we mean by this is that you could possibly have a computer which is totally isolated from all the other computers present in your LAN, in your network, you can install the required operating systems and all other software present in this special dedicated system and run your untrusted application over there, another option which is adopted quite often in cloud computing environments is to have Virtual Machines.

So with this solution you could create a virtual machine in your system, install a guest operating system in that virtual machine and then run the untrusted programs on that virtual machine, now a third solution is to use containers such as dockers which is not as strong as the air gapped systems and virtual machines but yet is able to provide some level of isolation in which you could run your untrusted applications.

So what we see is that all of these solutions the air gapped systems, Virtual Machines and containers would provide some degree of isolation in which untrusted application can execute however all of these are very coarse grain solutions, so what we mean by this is that each of this solutions would require a dedicated entity to be present for example with air gap systems, you would require a compete dedicated computer just to run the untrusted program.

With virtual machines you still would require an entire virtual machine to be present in your system while similarly with containers you would require a container such as docker to be present in your system, so what we would be seeing in this lecture is a fined grain solution which uses remote procedure calls or RPCs to create isolate entities.

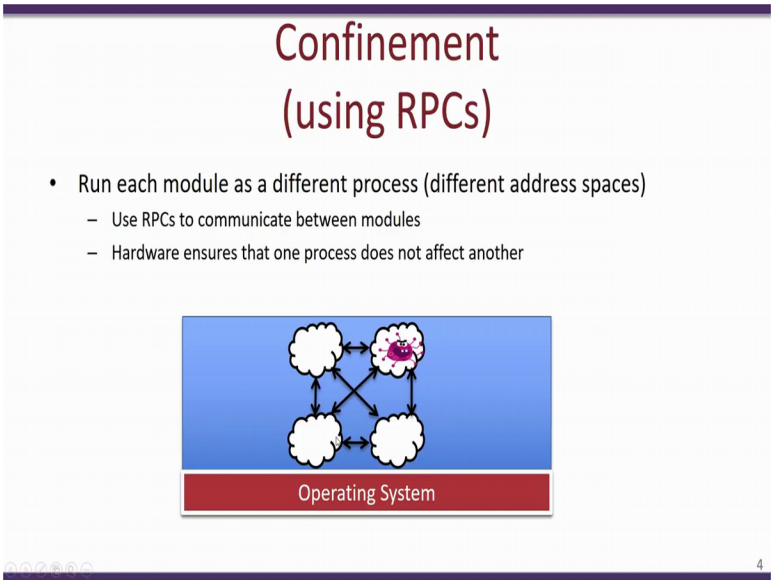
(Refer Slide Time: 04:18)



The scenario we consider in this lecture is as follows, let us say we have a large application running in a system so this application has several different models and all of these modules runs within a single application process, now let us say that there are several vulnerabilities in this particular application as we have seen therefore even if a single vulnerability gets exploited then the attacker would be able to get access to that particular application and therefore will gain access to the entire application space, thus what we need is to be able to design such an application so that even if the vulnerabilities are present in the application, the attacker will have limited amount of information that can be obtained from that exploited vulnerability.

So what we will look at in this lecture is how we design such large application so that even if a vulnerability gets exploited by an attacker, the amount of damage that can be caused by that exploit is limited and the entire application would should not be affected by that single exploit.

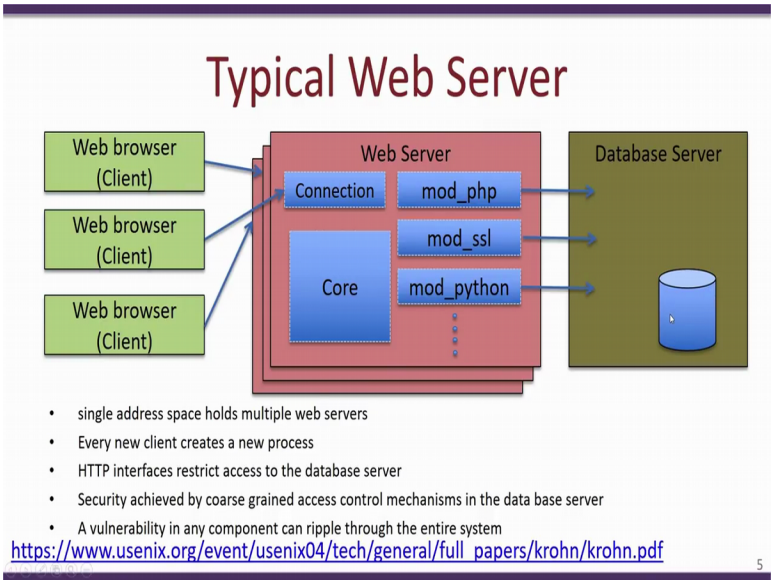
(Refer Slide Time: 05:40)



At a high level what we are going to do is that we are going to break this large application into several sub modules which would look something like this, each of this sub modules would then interact with each other using remote procedure calls or RPCs, now each of this sub modules runs as an independent process, therefore if a vulnerability is exploited by an attacker in one of these process modules it would mean that the attacker can only gain access to that particular module.

No other module such as these three would be compromised, thus they are able to contain the damage that is done to only a single module in the entire application.

(Refer Slide Time: 06:42)



As a case study for this approach we would look at this particular paper over here which designs something known as the OKWS web server, so it shows how a typical web server can be exploited and how a web server can be then designed so as to get higher levels of security, as a case study we would look at this particular paper on the OKWS web server, so this paper was published in 2004 and it starts off with actually redoing the Apache web server and all the vulnerabilities that were present in the Apache web server in 2004 and it also provides a design for a better approach for designing a web server.

So this web server was known as the OKWS web server, it should be note that although some of the aspects maybe different now, it still makes a very interesting case study so let us look how a typical web server look like in 2004, so the web server, a typical web server had several modules like this all of these modules ran in a single address space, so whenever a client connects to this web server through the web browser client, the connection module within its web server gets triggered and the connection module would then interpret details about the connection and then pass on that information to one of the different modules.

So for example if the web browser client has requested for a encrypted page then the connection module would pass that information to the SSL module which is present in the web server, on the other hand if the client requested for a PHP page then the connection module would send that request to the PHP module, further there would be one back end database which is accessed by

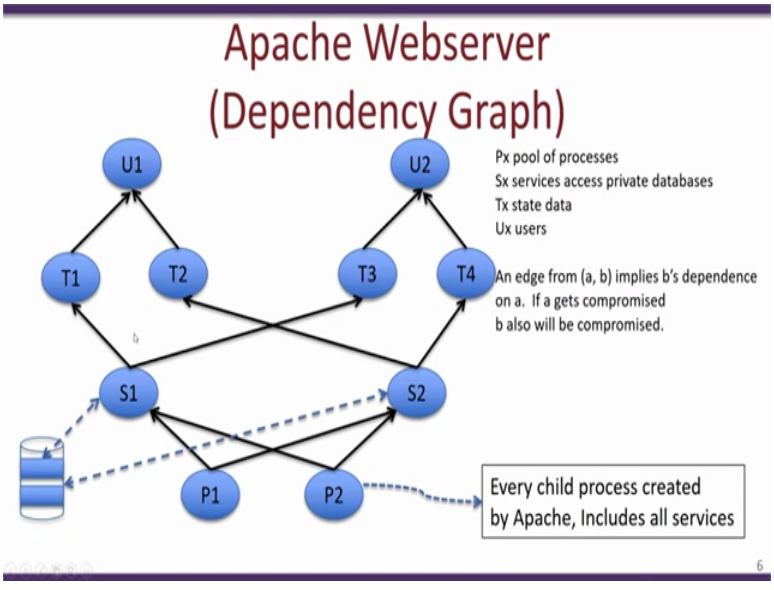
all the different modules present in the web server, so corresponding to that particular access, the response goes back to the client through the connection module.

Now two things to note over here is that all of these modules like the connection module, the core modules and the various other specific modules run from a single address space, that is all of them run as a single process, further if we have multiple clients connecting to this web server then there are multiple processes which responds each process amping a single corresponding client.

The back end database server however remains the same so now a single web server may host multiple web pages for example the single web server could host let us say a search engine as well as a daily newspaper, so both of these are finally managed by the single database server, now the security for this entire system rests on the database server, the isolation provided between the search engine webpage and the newspaper web page is only controlled by the coarse grained access control policies of the data base server.

Further, note that since each web server runs in a single address space, single vulnerability in any of these modules could compromise the entire web server and could possibly compromise the entire data base server as well.

(Refer Slide Time: 10:39)

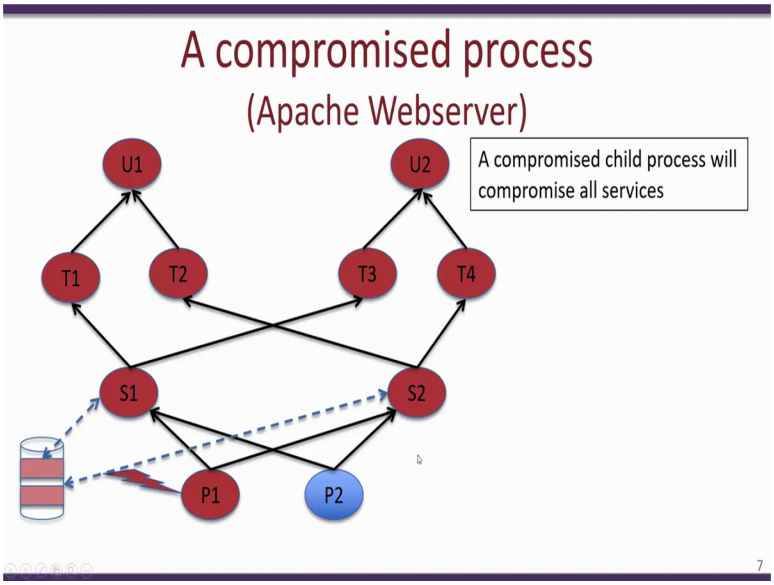


So what we have, let us say is two processes T1 and T2 and these two processes are created by two clients, user 1 and user 2, so let us say for example user 1 is connects to the web server and is using search engine which is hosted on the web server as we have discussed when the user 1 connects a new process gets created which is process P1 and based on the query from the user, the corresponding service in the data base is accessed, in this case this service would be the data base for the search engine and the result is then sent back to the user 1.

Now in a similar way let us say there is a user two and this user 2 connects to the same web server and is using the newspaper web page, now this particular graph over here is the dependency graph which shows how the various entities are dependent on each other, so for example each H over here would determine the dependence when the system gets compromised, example it process P1 gets compromised, then the H from T1 to S1 would imply that the service S1 would also get compromised.

Now note that due to the single address space that is present for the entire web server, so even though P1 is only meant to query the search engine due to the single address space the service S2 which corresponds to the newspaper is also compromised, similarly the other state data T1, T2, T3, T4 is also compromised. Let us say that there is a vulnerability in the web server as a result process P1 gets exploited, now we will trace the result of this single exploit.

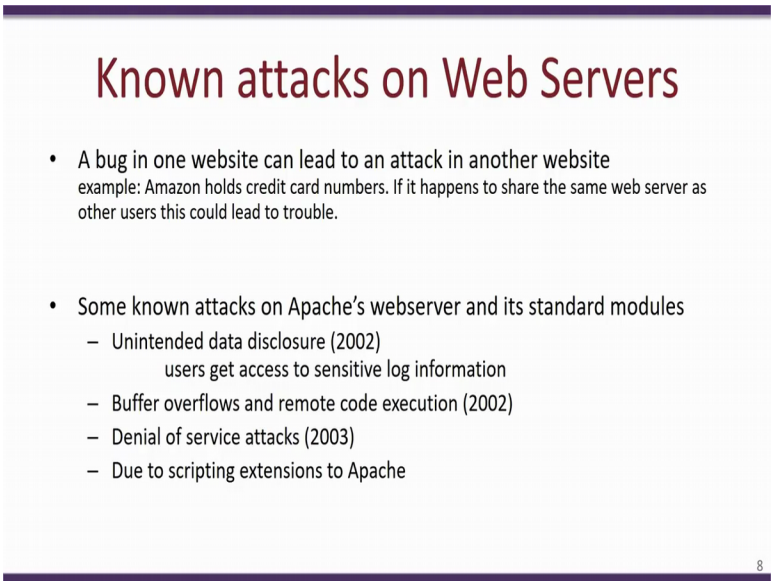
(Refer Slide Time: 12:42)



So the dependency graph now looks something this way so we have an exploit or a vulnerability that it exploit in the process P1 since service 1 and service 2 are connected to P1 therefore both the service 1 as well as service 2 are compromised, similarly since the service 1 as service 2 is compromised the entire data base is compromised.

Now the states T1, T2, T3 and T4 are compromised and therefore the user 1 and user 2 are compromised, so what we see from this dependency graph is that a single vulnerability in a particular process in the web server can compromise all users and all services that that web server supports.

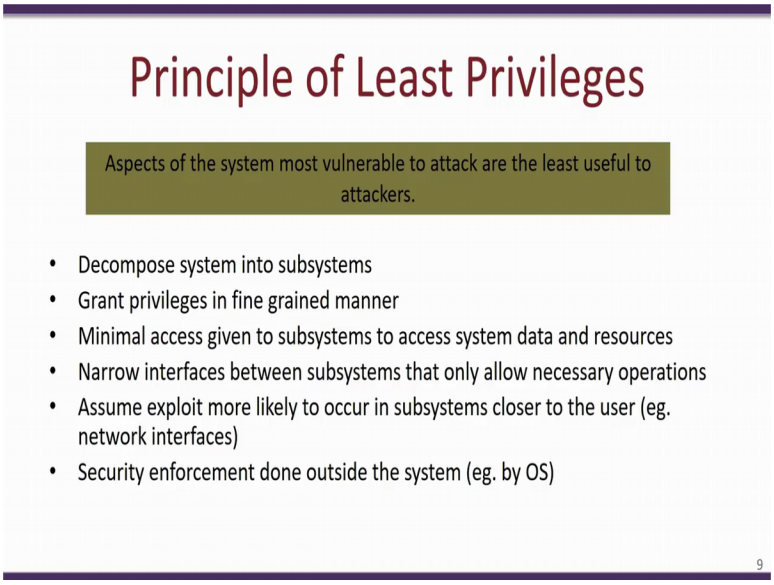
(Refer Slide Time: 13:36)



So due to this design architecture there have been seven different attacks on the Apache's web servers prior to 2004, so some of the attacks were unintended data disclosure in 2002 where users could get accessed to sensitive log information, buffer overflows were exploited in order to execute remote code, denial of service attacks were done, another scripting attacks were also on around the same time.

Now the OKWS web server which we will review today is based on something known as the principle of least privileges, so this is a very fundamental principle which covers a lot of security designs.

(Refer Slide Time: 14:21)



Principle of these privileges is based on the fact that aspects of the system most vulnerable to attack quite often have the least useful information for the attacker, so in order to design a system which complies with the principle of least privileges we should first decompose the system into several sub systems or several sub modules and run privileges in a fine grained manner essentially each of the sub modules should be given just limited amount of privileges so that it can execute.

So let us say for example we have a particular module which only accesses the USP device in your system, therefore your system should be designed in such a way so that this particular module is only given read and write access to the USB and is not able to access any other system resources, in a similar way such principles of least privileges can be applied to every module, another example is say for instance we have one particular module which only reads and writes from one file present in the system thus system should be defined so that even if there is a vulnerability in that particular module and the attacker is able to create an exploit and compromise that particular module.

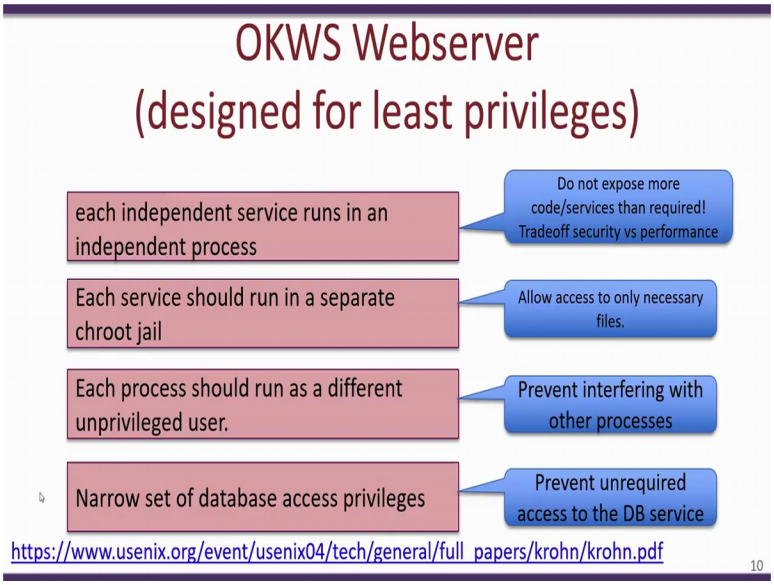
The only file that gets compromised there would be only one file that gets compromised, so in order to achieve this minimal access for a particular module the system should be defined so that there are narrow interfaces between the sub systems in order to achieve this minimal access between the sub systems what should be defined is that the interface between the various

modules should be as narrow as possible that is to say if there are two modules and one module just calls two functions in another module then the system should be defined in such a way that only those two functions can be evoked and no other function from that second module is accessible from the first module.

In addition to these other aspects that becomes crucial while designing such a secure system is that security should be enforced by someone outside the system for example the operating system which means to see that if you are designing a large application you should ensure that the security manager or the component which actually handles the security of that particular application is an entity which is outside that application and this is required because if that security module within the application itself gets compromised then the entire application itself is compromised.

On the other hand if the operating system itself is responsible for the application security then this operating system is outside the application, thus the attacker would need to compromise the operating system to gain access to the application and this could be several times more harder another very useful assumption is that during an attack scenario, it is more likely that attacks occur in interface which are closer to the user for example the network interfaces, so therefore comparatively it would be more critical to protect these types of interfaces.

(Refer Slide Time: 18:09)



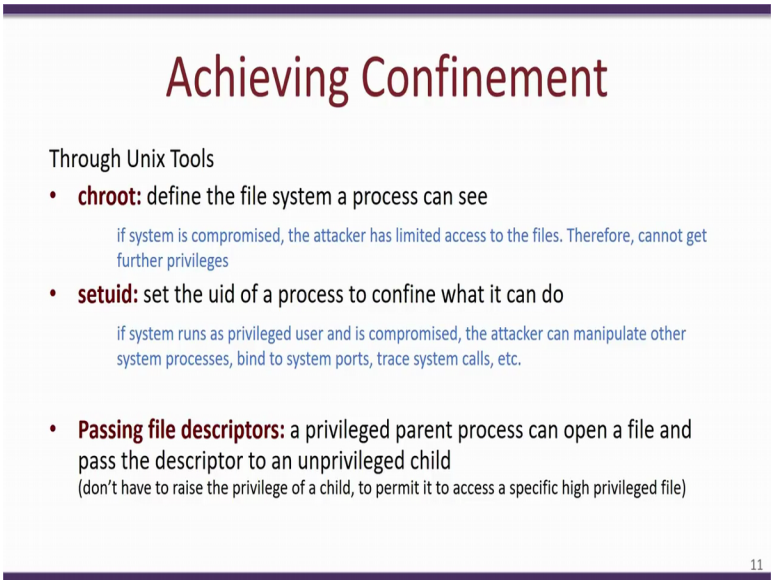
So now let us look at how the principle of least privileges is applied to the OKWS web server, more details about this thing can be obtained from this OKWS paper which is downloadable from this page, so in order to achieve the principle of least privileges, the OKWS web server is designed with certain rules, the first rule is that each independent service or each independent module runs in an independent process.

So for example if you have two different services so one for example is the search engine and the other one is say the daily newspaper, each of these should run in a independent process, by doing this we will not be exposing more quote than required, another principle that OKWS actually uses is that every service should run in a separate ch root jail or change root jail, so we will see more details of what ch root jail is but essentially what this particular point would provide is that a particular service would be able to access only the necessary files.

So a particular service for example would not be able to access some other file which is not required for its operation, third, every process would run as a different unprivileged user since the OKWS web server runs over a unique system, what is ensured is that every process and the process comprises of the various services would be running as different users thus we would be having a search engine run as a particular user as well as the daily newspaper run as a different user so by doing this we are able to further isolate one process from another, this is especially useful so as to prevent one process trying to debug the other process or trying to read the internal contents of the other process and so on.

The last principle over which OKWS web server is designed is that it provides a narrow set of database access privileges, by doing this the OKWS web server prevents unrequired access to the database service, before we go into details about how the OKWS web server uses each of these design rules we would require two fundamental aspects about unique systems.

(Refer Slide Time: 20:49)



So the OKWS server relies on the underlying unique system to provide the necessary security, three unique tools are used extensively during the design of the OKWS web server in order to achieve the required amount of isolation, so one is the change root, ch root and you can google or look up the man pages for change root and sect UID, so one is the change root which essentially defines the file system a process can see, the entire OKWS security is based on the security that the underlying unique operating system provides.

Essentially there are three very useful tools that is used quite often by the OKWS web server, so one is the change root, the second is the sect UID and the third as we have seen in our previous lecture is the use of passing file descriptors, the change root you can actually look up the man pages for this particular tool would define the file system for what a process can see by using the change root one can define a different file system for every process in the system thus what is possible by this is that every process would see a different file system.

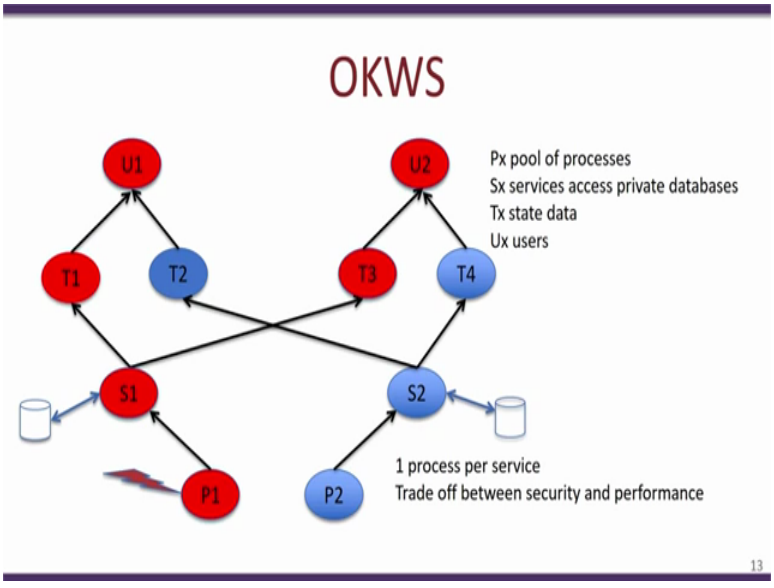
If a particular file system gets compromised and let say the attacker has been able to compromise that system and create a shell from that system, the attacker would be only able to see the files that is visible by that particular process, the files of other processes are completely invisible for the attacker, thus in other words the attacker has limited access to files and therefore cannot get further privileges and view other files which are present.

The second tool what we have seen in our previous lecture as well is that offset UID, using the set UID command, a particular process can be run using a specific user identifier or user ID using this what can be achieved is that a processes can be started and run as different unprivileged users, in OKWS each of the different processes are run as a different user in the OKWS web server each of the different process that gets started is run as a different user.

Thus if any of these processes gets compromised this particular process cannot debug or connect or use the internal contents of another process, further another useful application of set UID is that we could limit the privileges of a particular user, say for example if the system runs in privileged mode and is compromised, the attacker can manipulate other system resources like going to system ports etc.

Now using the setuid tool this thing can be prevented, the third utility which is exploited by OKWS is that in unique systems one process would actually create the file descriptors and could pass that file descriptors to other processes and once the second process obtains the file descriptors it would be able to access privileged files or any other files as required without any other checks. So using this we would be able to let a particular process access a privilege file without requiring to escalate a privilege of the child process itself.

(Refer Slide Time: 24:49)



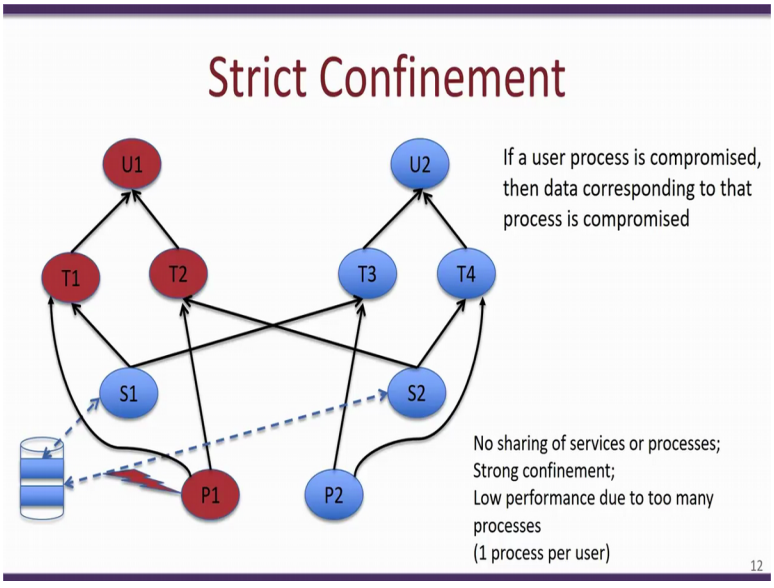
Now if we go back to the dependency graph and look at it with respect to how OKWS can get compromised we see that a lot less amount of components in the system are getting

compromised so with respect to the OKWS each process is tied with a service so this is quite unlike the Apache case where each user is tied with a specific process, with the OKWS we have a process tied to a service.

Therefore, if a process gets compromised then it is only that corresponding service that gets compromised and all users of that service would get hence compromised, thus we see that T1 and T3 is compromised, so let us say that service S1 corresponds to the web browsing while service S2 corresponds to the newspaper service, now let us say that service S1 corresponds to the search engine while service S2 corresponds to the daily newspaper service.

Now assuming that the search engine gets compromised then all the users and all the states corresponding to the search engine is compromised, on the other hand all the states and services corresponding to the daily newspaper is not compromised thus you see that process P2, S2, T2 and T4 which corresponds to the daily newspaper remains uncompromised while all the components corresponding to the search engine would get compromised thus you see that OKWS has been able to isolate one service from the other. While this is not the best way of achieving security, it is a good tradeoff between security achieved and the performance of the entire system.

(Refer Slide Time: 26:55)

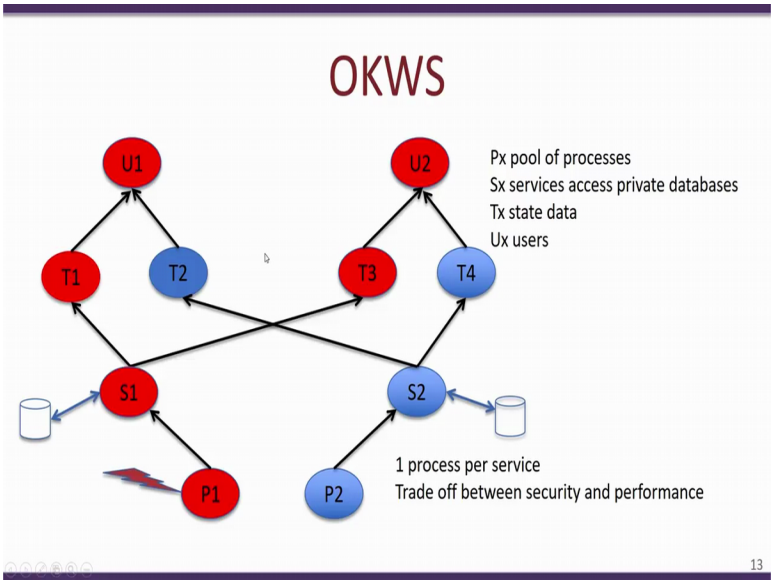


Now if one would actually design a system with strict confinement or strict isolation between the various entities it would look something like this, that is if a user gets compromised then only the activities corresponding to that user would get compromised.

So for example over here if U1 is compromised then the process P1 and the state data T1 and P2 is compromised, so nothing about the user 2, process 2 and so on is actually compromised, while this is the ideal situation, it is not very easy to implement, the reason being is that in order to implement this in practice we would require one process per user essentially with this particular model though it achieves maximum amount of security with the very strict isolation between the various entities.

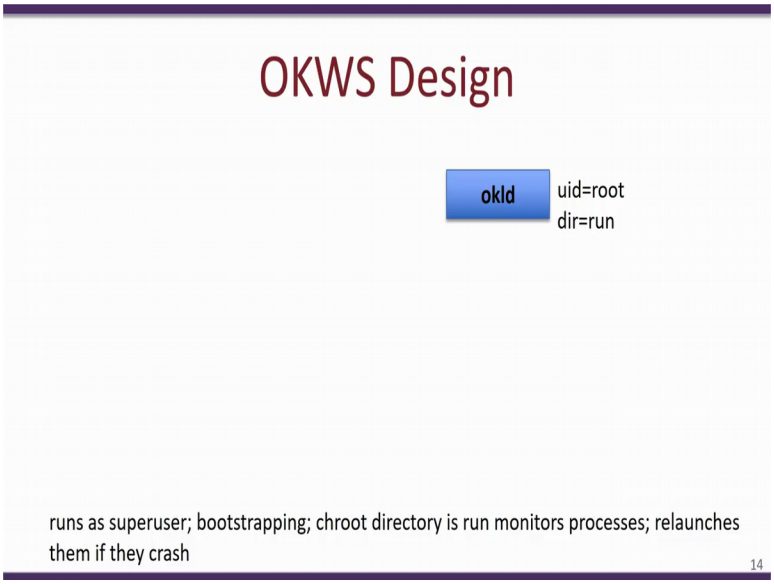
However, in practice implementing this type of design has got significant performance over heads, this is due to the fact that we have one process per user and if you have a large number of users connecting to the web server the number of process would increase many folds and therefore the performance will degrade.

(Refer Slide Time: 28:17)



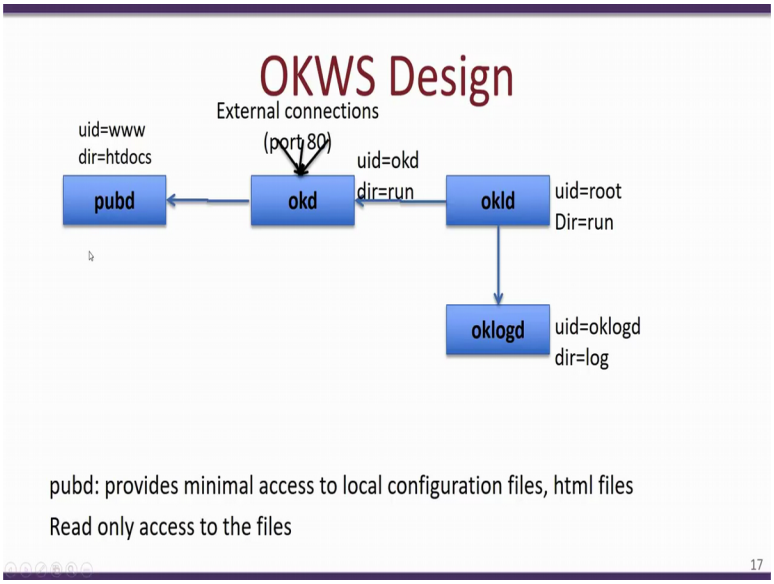
On the other hand, the OKWS web server design principle is able to create a compromise or a tradeoff between the security and the performance which is achieved.

(Refer Slide Time: 28:26)



So let us look a little more in detail about the OKWS design, now one of the main components in the design is this particular process known as the OKLD, so this particular process runs as a root user and it runs in a change root directory called run, this particular process takes care of bootstrapping the entire OKWS web server, it monitors the various processes which are executed and it determines if a particular process has crashed. If it has crashed then it would restart that particular process, after the OKLD process starts to execute, it would create two more processes.

(Refer Slide Time: 29:07)

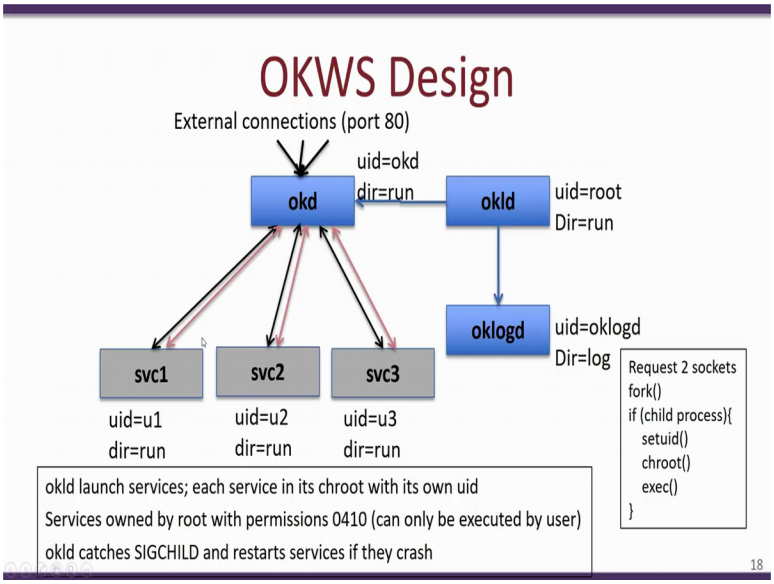


One is the OKD process and the OK log D process, the OK log D process runs with the user OK log D and in the change root directory called log, so this particular process takes a care of logging every activity of the web server, so in fact if other processes want to actually write to the file system or want to actually log something on the file system, it would actually communicate with the OK log D process through the RPC call.

Now the OKD process runs with the user OKD and in the file system run, so what this particular process does, it listens to external connections through port number 80 and then it forwards the request to specific services, so if the request is invalid then it sense a HTTP 404 error to the remote client if the request is broken then it could send an HTP 500 request to the remote client.

So another service which gets started is the pubd service, this runs with the user www and it runs in a change root directory called ht docs, so this pubd directory provides the minimal access to the local configuration files and other htms files so it has only read only access that is present, so one thing to actually note over here is that among the various processes it is only the OK log D process which is able to read and write to the corresponding file system, all other processes have only read access to the files.

(Refer Slide Time: 30:58)



Now once all of this is setup the OKLD would then generate the various services like the php service, the SSL service and so on, each of these services runs with a different root and the change root of run file system, now note that we can actually have all of these various processes

running from the change root file system because this run file system is ready only, so now there are connections between the OKD and the various services, for every connection that is requested the OKD would interpret the service that is requested and transfer that request to the corresponding service.

Side by side thus there is RPCs mechanisms between the OKD process and the various services so this RPCs are used to actually transfer request between the OKD and the corresponding services.

So in this way we have the various processes involved with the OKWS web server, the OKLD is kind of the root of this entire web server, it monitors and determines whether each of these services is running healthily, if it is not and if it detects that any of these services are stopped or has been crashed it then restarts that particular service, so this is how OKWS has efficiently been able to manage and isolate various modules within the large web server application.

So in the next lecture in the series, we will see how we would get finer grained isolation, now the over heads over here is that we have several processes that get involved and each process communicates with the other using RPCs, so these RPCs are quite heavy because each of them involve a system call and therefore have the operating system running, so n the next lecture what we would see is more fine grained isolation mechanisms where isolation is achieved within a single process, thank you.