Privilege Management: Lecture 8 (9/19/12)

Scribe: Rebecca Pottenger

September 29, 2012

"Extensible security architectures for Java" (Wallach et. al)

- This paper is not really about java: an excuse to talk about accessing control
- Start by telling us about access control in OS
- Subject wants to issue some request on object
- Reference monitor has job to inspect the request and see if it is authorized by system's security policy
- You break up the system into a bunch of subjects (users, or set of processes) and set of objects (all files on system, all hardware devices you might want to control access to) and make access control matrix
- What level of access does object have on subject (r, rw)
- Job is to check each request fits under policy specified in the access control matrix

This misses some interesting elements of the problem! Need to authenticate user (that is considered outside the scope)

- granularity: what is the granularity of subject or objects? In unix, it is users and groups. (subjects = users and groups, objects = every file)
- why did he say that a process is a subject? Because in unix a process runs with authorization depending on the user and not on the process (in general)
- all processes run under user id get rights associated with my id
- could have done finer grained thing where each process got its own set of rights, but finer grained policies are harder for system admins to specify (saw this in sandboxing where if you want policy for each pair of user and app have to specify lot more policies; coarser grained equivalence classes are easier because only have to specify for each equivalence class; but then lots of apps may get privileges you don't need)
- you can see the trade of space that isn't accounted for in our picture
- our picture is a static and simple view; set the policy and it is fixed, only have active and passive elements.

- Delegation changes things. He means something like: A makes request to B, for B to fulfill this request B must in tern make a request to perform some action on C. Have to think about privileges that A has, that B has, and figure out which are allowed and when they are allowed. B wants to do something to C on behalf of A. Do we use B's access control rules? Or A's, because B is doing it for A?
- We see this kind of question pop up in systems time and time again, Java paper has interesting view point about this question
- How does this get solved in the unix world? Let's say Alice wants to print a document, and she doesn't have access to the printer. Bob has access to the printer. How do you solve that problem? Alice emails Bob the document, and Bob prints it. This is an example of Bob using his access control rights to print something. Another way could get annoying if Alice wants to print multiple things. So Bob could give his password to Alice.
- System doesn't make it easy for Bob to just tell the system that Alice is ok.
- Bob could write a python script that will accept documents to be printed, will run under Bob's userid with his permission. That program will authenticate that it is Alice, and if it is then take what Alice supplied and print it (a proxy); automates what Bob would do
- We are working around the fact that system doesn't make it easy to do the delegation that we want to do
- Yet even this picture is kind of a lie! When Bob wants to print, Bob isn't really directly connected to printer. Bob doesn't key in the bits that are sent in to the printer, he uses some software to help him
- How would this work in unix? Start with root having access to printer. Root lets Bob to have access with the exact same delegation problem. So really it is with lpd. Bob runs lpr command, (which is not setuid, it runs with Bob's permissions, just a client) and that command makes some kind of communication to lpd (the service, which is running at root) and therefore has access to write to the printer; it can check the user id of the source of the request/command (which would be Bob); check if Bob is authorized to print, if he is then print (can check for quotas and whatnot).
- Even this is more of an oversimplification: really Bob is running some kind of a shell, types in a command to the shell which envokes lpr, etc.
- We talked about how you could implement this kind of a transition, where we have less rights to more rights. How can we do that in unix? If I want to write an lpd and lpr. Could make lpd setuid; communication and have lpd be a long running proxy (some kind of service that is always running)

    - Those two are almost equivalent at some conceptual level; setuid is on demand long running service

    - Setuid is almost an optimization, if you don't want lpd always running you can think of it as getting launched as needed (setuid says launch when needed, and launch as root)

- Build a table of a bunch of different systems in the world, to see what stance they take on the delegation question

- Unix Program Execution: A is running instance, B is underlying code on disk, received rights = rights that the running instance of B gets

    - So in the normal case, file (B) doesn't have any rights (not really a sensible question)

    - risk: try to lure you to run my malicious program

- Unix Setuid Exec: the received rights depends on what B does; if B wants it could get any of the rights of A and B

    - risks: if program does something dangerous, and you expose it when you shouldn't to S, could be giving Alice extra powers. Also, if you can change B on disk, you can do anything you want with permission T.

    - if program B has some security bug, then it might inadvertently expose all of the privileges T (let caller get access to all privileges T). summarize: might let A get more privileges than you intended (btw this is how jailbreaking works; take advantage of some vulnerability installed in setuid programs)

    - luring attack also susceptible, but not likely to happen in this scenario

- IPC = inter process communication. A is running program, B is something running background. A asks B to do something. Let's start with communication channel to local host.

    - Sender's rights are irrelevant; it's just a stream of bits. B decides what to do based on its own privileges T.

- Java Method Invocation (privileged): so now A and B are not processes; A is some code running (like some stack frame) and the rights of that running frame by default (if there are no delegations, just A making a request)

    - the way that works is Java looks at the code, what class that it is in, what security principle is associated with that class, based on what that's principle's rights are then that is the right the code gets.

    - So rights of A = what this code that is running could do; derived from java symantics of which class this code comes from (every class has set of rights associated with it)

    - So let's say code in A makes method call to code B, which is also in class and has its own set of rights.

    - What rights does B have while it is running, if it is called by A? the Java they have is based on stack inspection method. Paper says: you have any rights that B has, and you also have any rights that A has that B wasn't specifically denied (i.e. disable/enable privilege API's)

    - So in the most basic case, doesn't do the disable/enable business, so the answer is just the union of S and T

- O is the underlying java libraries/code that get invoked by B. they do some kind of stack inspection algorithm that determines whether to allow B's request. The algorithm from paper said that at this point, the implementation of the File library is going to call the security manager, which looks at the stack frame. So it sees the stack frame for B first, asks where the code comes from (i.e. the class) and checks out the security principle for class B. asks if it has ability to read the file that B asked to read. If allowed, then looks at the next stack frame which is method A, sees it is from class A. let's say A is applet we downloaded from the network; default policy is no access to anything on the file system. So the stackwalk algorithm will say method A doesn't have permission to read that file, and it doesn't so the request is denied. If you find a single one not allowed to make it, then everything is denied.

- What are consequences of this basic algorithm? It is super restrictive. Limits what kind of functionality you can write. You can't write a lpr (bob) : lpd (root) transition

- So what they do is, in cases where they need it, lets people elevate their privileges

- Enable privilege is the solution: It says that we don't want the intersection of S and T, we want T

- So instead they would see that B is allowed, B calls enable privilege, so the stack search stops, and B does its thing. So it allows the callee to select a different semantic for what gets transferred when you do a method request

Why do they have this crazy mechanism?

- Java stack inspection have these crazy semantics, different from everything else we've seen. Why?
- Well when java first came out, there was a tremendous fear of allowing stuff from outside to run on your computer. Had this threat model of some trusted code and some untrusted code.
- Is the java stack inspection model better/worse/similar to these other models? It is more fine grained; system will take away privileges for you as you go/run
- In java, you know that you are writing some part of code that might lead to privilege escalation, so you have to check code extra
- Unix is less susceptible. In the S intersection T case, not much opportunity for Alice to get privilege escalation; the loophole is that Bob can turn on user enabled privilege, and we are on the second line where privilege escalation is primary risk again
- Idea is to only turn on that row when the recipient opts into it, so hopefully used sparingly
- Potential hope of this architecture is we have reduced the risk of privileged escalation
- Luring attacks aren't possible in this setting. Not possible by Java method calls.

Suppose I'm writing a server side application:

- User : php application : database

- Which of our rows best fit this scenario? Usually just have a single database user that the php application connects to
- What are the security risks/where classify this?
- It's kind of like setuid row. Every access it makes to database is allow, entirely up to the code of the php to put in any restrictions
- Any bug in php, any logic that doesn't work in php app could allow one user to view another user's data when they aren't supposed to.
- How to reduce the likelihood of that kind of failure?
    - if you have database that only shows data to user. But that is only integrity not privacy
    - have small component to authenticate user, launch php protections with that specific user, kind of like each user has his own db
        - could have underlying framework of php track for each request wich authenticated user this is associated with; has some notion that for each user there is only a specific part of the db that user has access to
        - currently logged in user = principle, have database libraries check the rights associated with the user
        - could use the authentication mechanisms in the database itself. Map web user to db users and restrict access by db

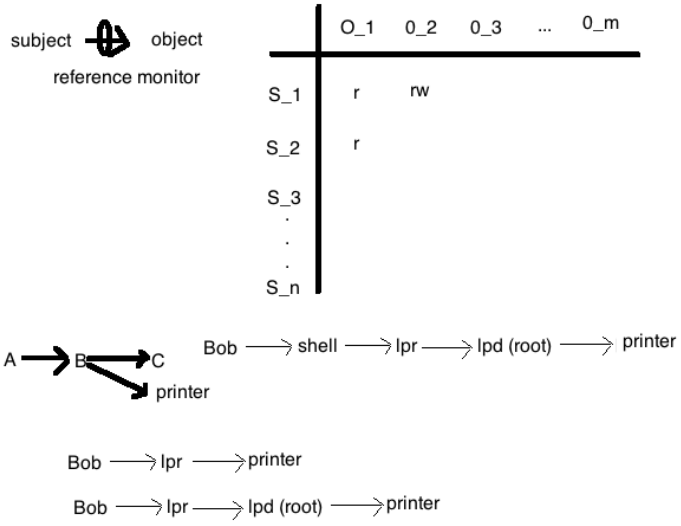
Suppose I'm working with android apps:

- App1 : app2 : OS
- One android sends an "intent" i.e. message that might start some code in 2nd app, which in turn has access to its own local storage/db/permissions (can place phone calls etc)
- How does the OS decide whether to allow this? The OS doesn't keep track of the chain of requests! It just looks at the app that is initiating the request, because this is just IPC.
- Security risk: if app2 has lots of permissions, then any bugs in it might allow app1 to do things that app2 can do even though app1 doesn't have permissions for that
    - Actually, ppl at Berkeley found that a lot of android apps have this vulnerability even preinstalled ones by Google
    - Combatted problem in a way that doesn't depend on every developer writing perfectly bug free code

The primary weakness in the java mechanism is that it relies on the security manager being able to look at the call stack

- A : B : C : ... : R

- Security manager wants to look at all of the parties that might have had some influence and make sure that they all had permission. Relies on that entire chain of influence being there in the calls tack.
- Any programming pattern that doesn't have entire chain of influence available doesn't get benefit of protection from security manager
- Most common case of that is even driven code: where A makes request to B, which in turn calls C but not right away; maybe gets put on a code and some separate thread goes through that invokes all of those, so security manager sees B and C etc were involved but not A



rights of A	rights of B	received rights	risks	system
S	irrelevant/none	S	luring attack	Unix program execution
S	T	T S U T on demand	privilege escalation luring	Unix setuid exec
S	T	T	privilege escalation	IPC
S	T	S intersect T T on demand	too restrictive privilege escalation	Java method invocation privileged version

A → B → O