

1/26 Lecture: Trust, open design, principles of secure system design

* (Black color is what the professor wrote on the board; Blue color is the notes from the class discussion. – Beili Wang)

Trust

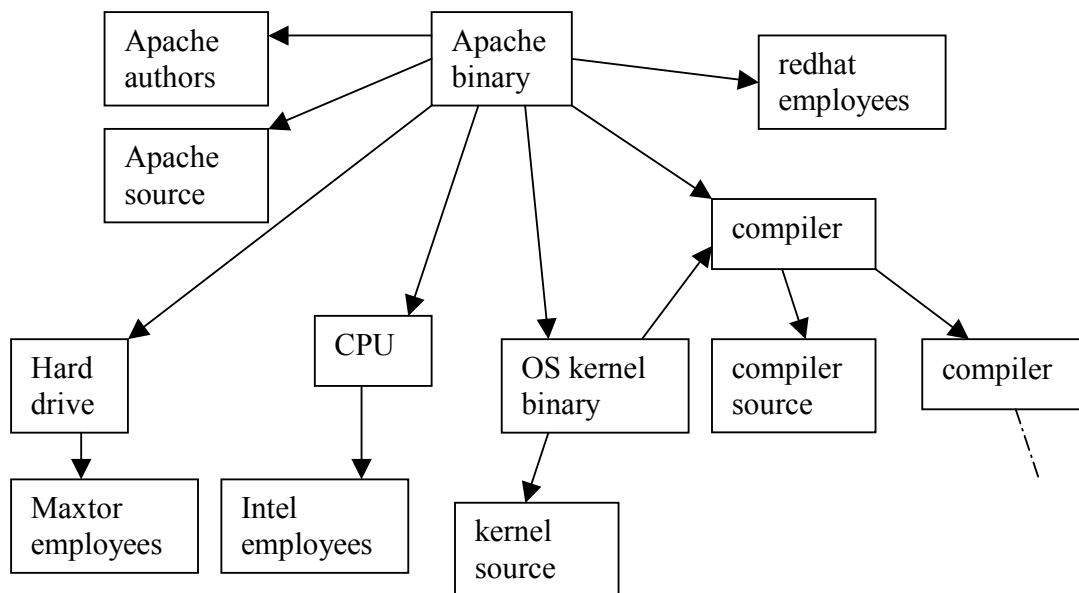
Trust == Dependence, BAD

Trust is transitive.

Trust in system security means dependence and dependence is bad.

Trust has transitive property. If A trusts B, B trusts C, it means A trusts C.

For Example: Apache enforce .htaccess. Apache enforce only .htaccess.



How to limit trust?

- verify everything
- multiple independent verifications (different people look at the stuffs)
- online testing (real time testing)
- build everything from scratch

Base case:

We lose if one system is compromised.

Independent checking:

Attacker must break all redundant checks.

For example: three organizations checking voting result or three compilers compiling the code.

Trusted Computing Base:

A resource is in the TCB if its correct operation is necessary for security.

You don't have to trust the code, if you don't install the code on the computer.

Rule: Keep TCB small.

Keep it small. Keep it simple.

Rule: Economy of mechanism.

- less code
- simple code

Clarity and conceptual simple

For example: Economy of mechanism

Exercise: passwords

Username unique

Hash password

- flat file – depends on OS
- encrypt
- table in database – depends on DB, OS

flat file vs. table in database

Storing in flat file is better but slow.

Rule: Failsafe Defaults.

- trust no one
- read only
- password protected
- default: DENY vs. ALLOW

if wrong:

break in (quietly) ALLOW – easy (ex: Microsoft)

complaints (noisily) DENY – forces administrator to set policy (ex: DOS)

default is ALLOW: if wrong, administrator knows when break in. The system failed quietly.

default is DENY: if wrong, administrator know when complaints. The system failed noisily.

Rule: When fail, fail loudly possible.

Rule: Least Privilege.

- Give each user/program least amount of power necessary to do its job.

For example: You are root or not.

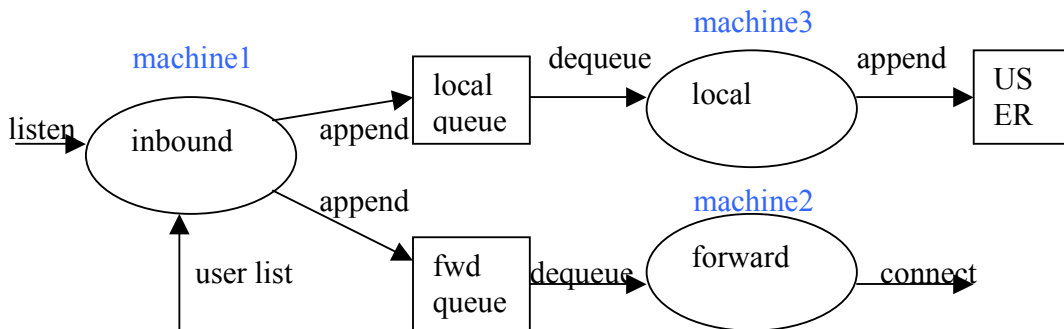
Root: can do anything.

Not Root: only do the things can.

Unix does not give good support of least privilege.

Exercise: Mail delivery system.

- listen to network
- look up users
- write to user mail
- queuing
- connect



does not have privilege to connect other machine
it is hard for machine1 connect to machine 2

Rule: Separation of Privilege

- Harder to subvert multiple subsystems.

For example: Force attacker to do two things instead one thing to win.

Rule: Least Shared Mechanism (Least Shared State)

e.g. Slashdot user account is different from slashdot's server user account.

- separate database
- log into the account does not have privilege to access the server.

Rule: Complete Mediation

- open
- creat

protect every way can get it.

For example: have security check on open, must have security check on create.

- open() vs. read()

For example: Unix enforce complete mediation on file system.

Process A

Process B

Open(f)

Chmod a -rf

Read(...)

Yes and No complete mediation

Unix design consider efficiency. It caching open and use for read. However, the caching result maybe stale. Unix consider this as ok.

Rule: Open Design

“Full Disclosure”

1. Application released
2. Bug found by user
3. user tells world (explore the bug) or 3'. user tells author in private
- 3.5. Hack, Hack, Hack
4. author fix bug
5. Users upgrade

For example: Paper Fuzz.

“Full Disclosure” is better. For example: Microsoft bugs will be fixed in average 50 days in full disclosure vs. 100 days without full disclosure.