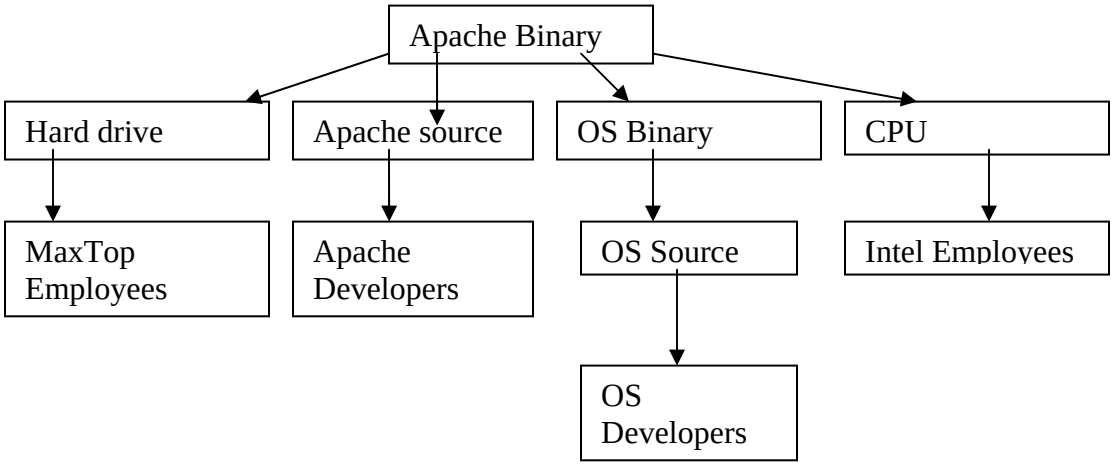
There are two ways in which security issues can be discussed:

Full Disclosure	Vs	Inform the Developer Only
Announce to the public eg. Treatises on Locks To shame the developers to take action		Inform the developers and wait for them to take action

In any case, trust is equivalent to dependence which is bad.
Problem: Transitive trust
eg. Apache enforces .htaccess to ensure correct access

Dependencies of trust:



What can we do to reduce trust/dependency:
1. Build it yourself
2. Policies (check the policies of any entity's security)
3. Prioritize
4. Redundancy (use things from multiple vendors; before: vulnerable to single failure after: vulnerable only to multiple failures)

Rules

- Keep the TCB (Trusted Computing Base) small

- Economy of Mechanism:
  Keep it simple, depend on a few things. A 100 line program is better than a 10000 line program. It has fewer bugs, easier to prove correct etc.
  Eg passwords in a flat file in Unix.

- Failsafe the Defaults:
  When no policies are defined,
  Default to deny Vs Default to allow
  Default allow requires specifying all the things that can go wrong.
  When default deny is wrong, it fails loudly (noticeable).
  Default allow fails silently.

- Least Privilege
  Give each process/user the least amount of power/access/privilege it needs to do its job.

- Separation of privilege
  Spilt a right among two users.

- Complete Mediation
  If you want to restrict access to an object, you must restrict every way of accessing that object.

- Least shared mechanism
  The key advantage of not sharing code is limited damage and less trust. But things are moving towards less data sharing and more code sharing nowadays.

- Psychological Acceptability (Unusable security is unused security)
  Eg Firewalls in corporate networks. Excessive security measures leads to leaks in them.