Salizer and Shroeder 1975 contains:

<u>Principles of Secure System Design</u>

- Principles of
  - Least Privilege
    A process will be given access rights to resources that only needed for this process to fulfill its function, which can minimize the possibility of system damage.
    - Bugs exist, deal with them by limiting application usage allowances. Don't let all users use all applications on the system.
      - Can be applied at the processes level- User X can only run applications W,Y,Z
      - Can also be applied to things such as programming languages
  - Fail-safe defaults(default deny)
    If a system is going to fail, it should deny access rights to an attacker, rather than letting the attacker in. “Default deny stance” means that everything that are not understood (permitted by the system) will be denied by default, which provides a great security.
      - Deny access if in doubt- We could be dealing with a simple bug or we could be under attack.
      - Bugs are usually found quickly- I'm logged in and should be able to access my email, yet the system is denying me access, probably a bug.
  - Economy of mechanism
    Design everything with simplicity in mind
    - *Simplicity yields assurance.* If our code gets too complex, more and more security holes will arise. Also, if the system is too complex, a simple bug or problem may be difficult to find and fix due to the size of the code.
  - Complete Mediation
    Don't allow attackers back doors or ways to bypass control mechanisms
    - Prevent this by checking *all* accesses to system resources
  - Open Design
    No security by obscurity
    - If we publish the source code, users will find and fix security problems as they arise.
      - *Bad idea...You cannot depend on the secrecy of your system design or source code to prevent attackers from attacking. This will not provide much security in the end.*
        - An obscure bug might not be found through common usage for example.
  - Separation of Privilege
    - Separation of duties gives security— suppose you have a lock and several distinct keys which are kept by different people, and the lock can be unlocked only if all the distinct keys are presented together. This scheme provides much more security than a lock with only one key. The same idea here also works on system security— a single compromised process (a single key) will not affect the security of the more important information (the lock).
  - Least Common Mechanism
    - Don't allow unnecessary sharing of resources
  - Psychological acceptability
    - If you design security/verification methods without the user in mind, even trusted users will find ways to subvert them due to difficulty of use

Trusted Path

Basically, it is a mechanism that ensures the integrity of communication channels.
Suppose there is a user login procedure, which can be viewed as a challenge-response protocol. It needs to make sure that this host-client communication channel is trusted. All processes other than login process will be shut to make sure that an attacker cannot spoof the user and get key strokes information from user.

*-At security critical operations (ex: login)*

- Make sure the communication channel between the system-user is *ultra* secure/assured
    - Don't allow lookalike GUI
        - GUI resembling windows login for example
            - Ctrl+Alt+Del to login is an example of how to address this issue
                - Locks out user processes and goes directly to the kernel.
                - Prevents programs from listening to key strokes

Spoofing UI's

Phishing attacks are common nowadays and they're hard to prevent. One reason is that if you are using your own work station, it is easy to distinguish which processes are malicious ones. However, when the system gets more and more complex, it is hard to tell which processes are bad. Malicious processes will also run under current user's id, so you only have to trust all applications that running on this machine.

- *Major concern, however not much has been done to address this issue*
    - Assumption has always been it's our system, we know what's on it!
        - Systems are becoming more and more complicated
            - We can no longer make this assumption
    - Creating lookalike GUI's is very easy
        - Absolutely *NO* OS level mechanisms to prevent login spoofing because there is no access control mechanisms for:
            - Dialog boxes
            - X-Window items
            - etc
- Hot area to get involved in?