UNIX Access Control

#access_control #access_control/unix #access_control/access_matrix

UNIX Access Control

- all objects are files
- classical protection system
  - limited access matrix
  - discretionary protection state operations
- practical model for end users
  - involves some policy specification
- **mode bits** - first column in `ls -al`
  - defines read, write, and execute for each user group
  - extra flag if file is directory
- example access matrix
  - suppose private key file for subject J is object O_1
    - only J can read
  - suppose public key file for J is object O_2
    - all can read but only J can modify
  - suppose all can read and write from object O_3
  - resulting access matrix

	O_1	O_2	O_3
J	R	RW	RW
S_2	-	R	RW
S_3	-	R	RW

- questions to consider for example access matrix
  - **secrecy** - does the protection state for entry J, O_1 ensure the secrecy of J's private key file, O_1?
  - **integrity** - does the protection state for entry J, O_2 protect the integrity of J's public key file, O_2?
  - **trusted processes** - does it matter if we do not *trust* some of J's processes?
    - yes it does

- *trojan horse* - attacker-controlled code run by J can violate secrecy
  - J's row
- *confused deputy* - attacker may trick untrusted code run by J to violate integrity
  - O_2's column

- **confused deputy** - having a subject with read and write privileges on all files write corrupted information to a predicted file
  - example - server handles requests for functions to process a file received by a client
    - client sends name of file
    - server computes function on the file
    - server writes information from the function to a specified file (e.g. billing.txt)
    - client cannot write billing.txt, but the server has read/write privileges on all files

Protection vs Security

- **protection** - security goals must be met under *trusted* processes in order to achieve
  - protects against an error by a non-malicious entity
- **security** - security goals must be met under *potentially malicious* processes in order to achieve
  - protects against any malicious entity
    - example - for J, non-malicious processes should not leak the private key by writing it to O_3