Computer Security - CheatSheet

IN BA5 - Thomas Bourgeat

Notes by Ali EL AZDI

October 26th, 2025

CompSec Properties

- **Confidentiality**. prevention of unauthorized disclosure of information. *authorized users may read a file*
- **Integrity.** prevention of unauthorized modification of information. *authorized programs may write a file*
- **Availability.** prevention of unauthorized denial of service or access to information and resources.
  authorized services can access a file
- **Authenticity.** assurance that entities (users, systems, or data) are genuine and can be verified as such.
- **Anonymity.** protection of an individual's identity from being disclosed or linked to specific actions or data.
- **Non-repudiation.** assurance that a party in a communication cannot deny the authenticity of their signature or the sending of a message.

The Adversary. malicious entity aiming at breaching the security policy and **will** choose the optimal way to use its ressources to mount an attack that violates the security properties.

Threat Model. describes the ressources available to the adversary and their capabilities *(has access to internet, but doesn't have access to the internal network of the company.)*

Threat. Who might attack which assets, using what resources, with what goal, how, and with what probability

Vulnerability. Specific weakness that could be exploited by adversaries with interest in a lot of different assets *(API is not protected, password appears in plain text. . .)*

Harm. The bad thing that could happen when the **threat** materializes. *(adversary steals the money, learns my password. . .)*

Security Policy. high level description of the security properties that must hold in the system in relation to assets and principals.

- **Assets (objects).** anything with value (data, files, memory) that needs protection.
- **Principals (subjects).** people, computer programs, services

Security Mechanism. Technical mechanism used to ensure that the security policy is not violated by an adversary within the threat model, **we can only prepare for threats we're aware of**

(Policy. ensure messages cannot be read by anyone but the sender and the receiver, Mechanism. encrypt the message before sending)

Composition of Security Mechanisms

- **Defence in depth.** As long as one remains unbroken the Security Policy isn't broken) *(two-factor auth)*
- **Weakest Link.** if anyone fails the Security Policy, it is broken. *(security questions for a lost password, just need to know the answer. . . .)*

Humans can be vulnerabilities - phishing attacks, bad use of passwords. . .)

To show a system is secure. (under a *specific* threat model)

Attacker - Just one way to violate **one** security property is enough.

Defender - No adversary strategy can violate the security policy.

Security Argument. Rigorous argument that the security mechanisms in place are indeed effective in maintaining the security policy subject to the assumptions of the Threat Model.

Principles of CompSec.

1. **Economy of mechanism**
   Keep the security mechanism/implementation design as simple and small as possible. Why ?
   a. Easier to audit and verify.
   b. Testing is not appropriate to evaluate security.
   Trusted Computing Base (TCB).
   Every component of the system on which the security policy relies upo hardware, firmware, software.
   The TCB is trusted to operate correctly for the security policy to hold. → If something goes wrong in it, the security policy may be violated
   It **must** be kept small to ease verification (economy of mechanism) and diminish the attack surface
2. **Fail-safe defaults.** Base access decisions on permission rather than exclusion. *(Whitelist, do not blacklist)*
   If something fails, be as secure as it does not fail errors / uncertainty should error on the side of the security policy. **Don't** try to fix on error! (*Automated doors: if they cannot close, stay open*)
3. **Complete mediation. Every** access to every object must be checked for authority A Reference Monitor mediates all actions from subjects on objects and ensures they are according to the policy. Tradeoff time_to_check vs. time_to_use
4. **Open design** The design should not be secret
   – Always design as if the enemy knows the system.
   – When you design...
     ∗ Crypto. Only keep the key secret
     ∗ Authentication. Only keep the password secret
     ∗ Obfuscation. Only keep the used noise secret
   assuming the thread model can't get a hold of the system is unrealistic (employee corruption, ...)
5. **Separation of privilege.** No single accident, deception, or breach of trust is sufficient to compromise the protected information
   – **Privilege.** A privilege allows a user to perform an action on a computer system that may have security consequences. *(create a file in a directory, access a device, write to a socket for communicating over the internet. . .)*
6. **Least Privilege.**
   Every program and every user of the system should operate using the least set of privileges necessary to complete the job. *Rights are added as needed, discarded after use. Users should get to know about things if they **have** to.*
7. **Least Common Mechanism**
   Minimize the amount of mechanism common to more than one user and depended on by all users. Every shared mechanism represents a potential information path between users.
8. **Psychological acceptability** It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. *(hide complexity, keep ressources as accessible as before,mental model of users must match security policy/mechanisms, cultural acceptability. . .)*
9. **Work Factor**
   Compare the cost of breaking the mechanism with the resources of a potential attacker. *(cost of compromising insiders, cost of finding a bug, monetization. . .)*
10. **Compromise recording** Reliably record that a compromise of information has occurred [...] in place of more elaborate mechanisms that completely prevent loss. (*keep tamper-evidence logs, what if you cannot recover them? (confidentiality), how to keep integrity ? (Blockchain), logs may be a vulnerability ? (Privacy), logging the log ? (Availability). . .*)

Access Control. Security mechanism that ensures all accesses and actions on objects by principals are within the security policy.

no *chicken soup (checks everywhere in code), use a reference monitor, module used all over code checking for **subjects** and **actions***

Discretionary Access Control (DAC). Object owners assign permissions, ownership of resources. *Linux, Social Networks*

Mandatory Access Control (MAC). Central security policy assigns permissions, usually for organizations with need for central control. *Military, Hospital Environments, Banking.*

Access Control Matrix. abstract representation of all permitted triplets of (subject, object, access right) within a system.

	file1	**file2**	**file3**
Alice	read, write		read
Bob		read, write	read, write

Complexity. $O(f \cdot u)$

User & Group Identities.

Most modern systems rely on DAC.

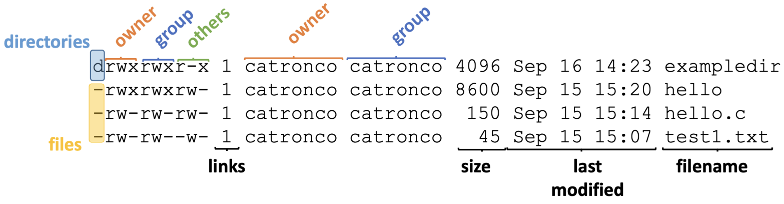
UIDs / GIDs. numerical identifiers for users and groups.

/etc/passwd: `username:password:UID:GID:info:home:shell`

/etc/group: defines secondary groups.

Each user has a home directory and belongs to one or more groups.

UNIX Model. Everything is a file.



```
drwxrwxr-x 1 catronco catronco 4096 Sep 16 14:23 exampledir
-rwxrwxrw- 1 catronco catronco 8600 Sep 15 15:20 hello
-rw-rw-rw- 1 catronco catronco  150 Sep 15 15:14 hello.c
-rw-rw--w- 1 catronco catronco   45 Sep 15 15:07 test1.txt
```

Directories.

Read → list files; Write → add/remove files; Exec → traverse (cd).

Access Control List (ACL). associate permissions to objects, stores permissions close to the resource.

file1: (Alice,read/write), file2: (Bob, read/write),
file3: (Alice,read),(Bob,read/write)

⊕ easy to determine who can access a resource and to revoke rights by resource.

⊖ difficult to check all users rights, to remove all permissions for a user. it's also difficult to delegate perms.

Role Based Access Control (RBAC). access granted based on user roles and predefined permissions. systems have too many subjects (that come and go) → large dynamic ACLs.

Subjects are often similar to each other and get assigned the same rights.

1. Assign permissions to roles, 2. Assign roles to subjects
3. Subjects have the permissions of their assigned role

⊖ role explosion (temptation to create fine grained roles), limited expressiveness, difficult to implement separation of privilege.

Group Based Access Control (GBAC). access granted based on user group membership. **exactly like RBAC but instead of roles, permissions are assigned to groups.** groups are (typically) broader, less specialized, often representing organizational units rather than specific functions.

1. Assign permissions to groups
2. Assign subjects to groups
3. Subjects have the combined permissions of all their groups

⊖ coarse granularity, overlapping group memberships, inconsistent permissions, difficult to manage if users belong to many groups.

*In case of **Negative Permissions** check negative permissions first before group, think of system crashes before checking negatives.*

Capabilities. associate permissions to *subjects*, stores permissions close to the user/process.

Alice: {(file1, read/write), (file3, read)}
Bob: {(file2, read/write), (file3, read/write)}

⊕ easy to determine all permissions of a user and to delegate rights by subject.

⊖ difficult to determine who can access a given resource, and to revoke rights by resource.

Ambient Authority. an action succeeds if the subject only specifies the *operation* and the *object name*, not the specific authority used.

⊖ leads to accidental misuse of authority, programs may act with more rights than intended. → Confused Deputy Problem.

Confused Deputy Problem. when a program (deputy) is tricked into misusing its authority on behalf of another subject.

Alice runs compiler(input, bill) ⇒ compiler (with write access to bill) overwrites billing file. ⇒ Alice uses compiler's authority to modify bill indirectly.

⊖ ambient authority allows unintended privilege use.

Solutions:

1. Restrict privileged process access.
2. Make privileged process check user's authorization.
3. Use **Capabilities** to explicitly delegate rights.

Permission check order:

1. If process UID = file owner → check owner bits.
2. Else if GID matches → check group bits.
3. Else → check "other" bits.

Root (UID 0) bypasses all checks.

Changing Permissions.

chmod. modify permission bits. `chmod +r file`, `chmod 666 file`.

chown. change file owner/group(opt.) `chown root:staff /srv/config`

chgrp. change file group. `chgrp www-data /var/www`

Special Rights.

suid (set user ID). run program with owner's privileges.

`chmod u+s filename - rws r-x r-x 1 root root /bin/passwd`

→ allows normal users to change passwords without full root access.

⊖ dangerous if exploited, in TCB.

sgid (set group ID). run program with group's privileges.

`d rwx r-s r-x 2 root staff /srv/shared`

sticky bit. on directories, prevents deleting/renaming files you don't own (e.g. `/tmp`).

`d rwx rwx rwt 10 root tmp /tmp`

Special Users.

root: UID 0, full privileges, bypasses checks, in TCB.

nobody: UID -2, owns no files, minimal privileges, safer for untrusted code.

⊕ flexible, simple model, widely adopted.

⊖ relies on **ambient authority**, prone to Confused Deputy attacks.

Windows: DACL. Controls access to objects via list of ACEs (Access Control Entries).

ACE. `<Type, Principal, Permissions, Flags>`.

- **Types.** `Allow / Deny` (negative / positive). Deny takes precedence and ordered with Denied permissions first.
- **Principal.** User or group (SID).
- **Permissions.** Fine-grained rights*(Read, Write, Execute, Delete...)*
- **Flags.** Inheritance, propagation, object-specific.

Access Tokens. Thread/process carries user + group SIDs checked against DACL.

Least Privilege. Users run limited by default; elevation via "Run as administrator"

Security Model. a design pattern for a specific security property or set of properties. (⚠not covered by model - who are the subjects? what are the objects? what mechanisms to use to implement it?)

Bell-La Padula (BLP). security model where Subjects S and objects O are associated to a level of Confidentiality.

- Access rights. Execute, Read, Append, Write.
- Objects are associated to a Security Level = (Classification, set of categories).

{(Unclassified < Confidentiality < Secret < Top Secret), {NATO, Crypto, Nuclear}}

Dominance Relationship. Transitive, There always is a Top and Bottom, Only partial ordering (some pairs of elements can't be compared).

A security level (l_1, c_1) dominates (l_2, c_2) if and only if $l_2 \leq l_1$ and $c_2 \subseteq c_1$.

eg. (*Secret*, {*Nuclear, Army*}) ≥ (*Confidential*, {*Army*})

- **Clearance.** max security level a subject has been assigned. *clearance-level(S_i).*
- **Current Security level.** subjects can operate at lower security levels. *current-level(S_i).*

BLP to create Confidentiality Policies.

Properties:

Simple Security Property. if (subject, object, w/r) is a current access, $\Longrightarrow$ level(subject) dominates level(object). **(SUBJECTS CAN'T READ UP)**

Star Property. If a subject has simultaneous "observe" (r,w) access O_1 and "alter" (a, w) access to O_2 then level O_2 dominates level O_1.

(SUBJECTS CAN'T WRITE DOWN). *changing object's perms is write-like.*

Discretionary Property. A subject may only access an object if it has explicit permission for that specific type of access (r, w, x, etc.) defined by the access control matrix.

Basic Security Theorem(induction). if all state transitions are secure, and the initial state is secure, then every subsequent state is secure regardless of the input.

Covert Channels. Any channel that allows information flows contrary to the security policy, *Storage channels(shared counters, ...), Timing channels(queuing time...).* ⚠Least Common Mechanism.

can be mitigated by adding noise or isolation(no high ↔ level communication).