Computer Security - CheatSheet

IN BA5 - Thomas Bourgeat

Notes by Ali EL AZDI

October 26th, 2025

CompSec Properties - **Confidentiality**. prevent unauthorized disclosure of information. *authorized users may read a file*
- **Integrity.** prevent unauthorized modification of information. *authorized programs may write a file*
- **Availability.** prevent 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 adversaries could exploit with interest in a lot of 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) needing 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 security mechanisms in place are effective in maintaining security policy subject to assumptions on Threat Model.

Principles of CompSec.
1. **Economy of mechanism** Keep security mechanism design simple and small ⇒ Easier to audit and verify, testing is **not** appropriate to evaluate security.
- **Trusted Computing Base (TCB).** Every component of the system on which the security policy relies upon *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
must be kept small to easy verification / 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.
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. ⚠ *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. (for Crypto, only secret is key. for Authentication, Only secret is password. for Obfuscation, Only used secret is noise.).* **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...)*
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 on need, discarded after use. Users should only 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, 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** Detect and record security breaches with tamper-evident logs, ensuring traceability, integrity, confidentiality, and availability of recorded data.

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,....*
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. $O(f \cdot u)$
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.
⊖ hard to check all users rights, to remove all perms. for a user, 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.*
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.

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.

directories owner group others owner group

```
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
```

files links size last modified filename

Directories. Read → list files; Write → add/remove files; Exec → traverse (cd).
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. *puts s in owner's x field*
`chmod u+s filename`
allows normal users to change passwords without full root access.
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.
`d rwx rwx rwt 10 root tmp /tmp`[1px] **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.
ACE (Acces Control Entry). `<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 admin."

Capabilities. associate permissions to *subjects*
Alice: {(file1, read/write)} Bob: {(file2, read/write), (file3, read/write)}
⊕ easy to determine all permissions of a user and to delegate rights by subject.
⊖ hard to determine who can access a resource or to revoke rights by resource.

Security Model. a design pattern for a 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.

Simple Security Property.
if (subject, object, w/r) is a current access, $\implies$ 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).

BIBA (Integrity). Security model where subjects S and objects O are associated to a level of **Integrity** (trustworthiness). Access rights. Read (Observe), Write (Modify)

BIBA to create Integrity Policies.

Simple Integrity Property.
If (s, o, r) is a current access, then level(s) dominates level(o).
(SUBJECTS CAN'T READ DOWN) ⇒ prevents contamination from low to high.

∗-Integrity Property.
If (s, o, w) is a current access, then level(o) *does not* dominate level(s).
(SUBJECTS CAN'T WRITE UP) ⇒ prevents low from corrupting high state.

Discretionary Property (DAC). Subject must still have explicit permission for the access (r, w, x) per the access control matrix.

Basic Integrity Theorem (induction). If the initial state is integrity-secure and all state transitions satisfy BIBA properties, then all reachable states preserve integrity.

BIBA Low-Water-Mark Variants (taint-tracking).
1. *For Subjects.* Subjects start at their max integrity.
On read, current-level(s) = min(current-level(s), level(o)).
⇒ once tainted, subject sinks; prevents writing up thereafter.
2. *For Objects.* On write by s, level(o) = min(level(o), level(s)).
⇒ objects "sink" easily; can cause integrity collapse.
Mitigation: replicate high/low objects; sanitize before promotion; detect/flag unexpected level drops.

Invocation Controls.
Simple Invocation. only allow subjects to invoke subjects with a label they dominate.
⊕ protect high integrity data from misuses by low integrity principals
⊖ what level is the output ?
Controlled Invocation. Only allow subjects to invoke subjects that dominate them. ⊕ prevents corruption of high integrity data
⊖ hard to detect polluting information.

Sanitization (lifting low → high).
- *Fail-safe default:* deny by default; elevate only after checks pass.
- *Whitelist over blacklist:* validate that *all* required properties of "good" hold.
- *Context-aware:* encoding, schema, range, semantics, and provenance checks.
Note: Sanitization bugs commonly break integrity guarantees.

Supporting Principles.
- **Separation of Duties:** split critical actions across principals.
- **Rotation of Duties:** limit tenure and constrain concurrent roles.
- **Secure Logging:** tamper-evident, consistent logs for detection/recovery.

Chinese Wall Model.
All objects are associated with a label denoting their origin.
(Pepsi, Coca-Cola, Microsoft Audit, Microsoft Investments)
Define **conflict sets** of labels. {*Pepsi, Coca-Cola*}, {*Microsoft Audit, Microsoft Investments*}.
Subjects are associated with a **history** of their accesses to objects and their labels.

Access Rules.
A subject can access an object (read or write) **only if** the access does not allow information flow between items with labels in the same conflict set.

Example (Direct Flow).	**Example (Indirect Flow).**
1. Access to Pepsi (*OK*)	1. Alice accesses Pepsi
2. Access to Microsoft Invest (*OK*)	2. Bob accesses Coca-Cola and IBM
3. Access to Coca-Cola (*Denied*)	3. Alice tries to access IBM (*Denied.indirect link via Bob*)

Sanitization.
Allows more flexibility by "un-labeling" some items, enabling controlled sharing or reuse of data.

Cryptography.
Data in transit. Securing communications.
Data at rest. Securing stored informations
Let C the Ciphertext, K the Key, M the Message/Plaintext
$C = E_K(M)$ and $M = D_K(C)$

Keyspace.
The total number of possible keys that can be used in an encryption algorithm.

Invertibility Requirment.
$\forall K, M, D_K(E_K(M)) = M$ (otherwise message can't be recovered)

Security Requirement. Functions should be hard to invert without knowing K.
Ideal Case - adversary must try every possible combination of keys (bruteforce).

Caesar's Cipher
- Encryption. Shift each letter by a fixed number (K)
- Decryption. Shift each letter by a fixed number (-K)
Keyspace. Only 25 possible keys.
So $\log_2(25) = 4.6$ bits of security, too small for real-world use.

The Substitution Cipher
- Each Letter is mapped to a unique, different letter, defined by a permutation of a 26 letters alphabet.
Keyspace. $26! \approxeq 4.03 * 10^{26}$
Both can be broken using **Frequency Analysis Attack**.

Frequency Analysis. Use statistical properties of the language.
1. Most frequent letter in English is 'e'. 2. Identify most frequent letter in Ciphertext.
3. Map it to 'e'.

Ideally, An N-bit key should offer security as close to N bits as possible (require 2^n attempts), if not the algorithm is considered **broken**.

Types of Adversaries. security models
- Passive Eavesdropper - adversary can only read the ciphertext
- Active Attacker - adversary can influence the system *(corruption of one of parties, ...)*

Known Plaintext Attack.(KPA) Active model where the attacker is given access to multiple pairs (M, C) corresponding to messages and their corresponding ciphertext, all encrypted with a secret K.
From these pairs, she tries to guess key K to decrypt other messages *realistic because of message headers,*

Chosen Plaintext Attack (CPA):
- Suppose Eve convinces Alice to encrypt chosen messages with the secret key
- $\forall m$, Eve gets access to $E_K(m)$. Eve has access to an Encryption Oracle.
(realistic - suppose she gets access to an encrypted messaging app for limited time, or to an encryption api).
very broken if Eve chooses to encrypt entire alphahabet, revealing the key instantly.

Side-Channels Attacks
- Timing attack - Eve measure how long it takes for a given message to get encrypted or a ciphertext to be decrypted
- Power Analysis - Eve observes the energy consumed by the device doing the crypto.

One-Time-Pad (OTP).
Goal - Remove frequency analysis
- Use a key of random bits as along as the message.
- $Enc(k, m) = m \oplus k$ - $Dec(k, m) = Enc(k, m) \oplus k$
Key should be random for every sent message. Otherwise attacker can collect information about the message.

OTP is technically **Perfectly Secure**.
- For any ciphertext C, every possible plaintext M is equally likely
- The key K is uniformly random, so C gives the adversary zero new information about M.
Formally (Perfect Secrecy): $\forall m, c, P(M = m | E_k(m) = c) = P(M = m)$.
Guarantees **confidentiality** ⊖ message-length/used once keys.
Integrity Attack Example (Eve flips the first bit of M):
1. Eve flips the first of C to get C'
2. Bob decrypts C': $M' = C' \oplus K = (M \oplus K \oplus \Delta) \oplus K = M \oplus K$

Symmetric Cryptography Schemes Encryption and decryption done with the **same key**
- **Block Ciphers.** Operate on fixed-size blocks *(128bits)*.
AES (Advanced Encryption Standard)
- **Stream Ciphers.** Operate on bit/byte at a time, like pseudo-OTP.

Symmetric Cryptographic key
- **Known to both parties.**
  Partners must agree on the key before starting using the primitive
- **Reused.**
  The keys is pre-shared once and then reused *(but keys have a "duration")*
- **Must be secret.**
  Reveling the key eliminates any protection provided by the primitive

Stream Ciphers - The pseudo-OTP. emulate OTP while solving key-length problem.
1.Shared secret short Key (K)
2. Key Stream Generator - Uses K and an Initialization Vector (IV) to produce an arbitrary long, pseudo-random bit stream (S).
(generator needs a key as main seed to generate a predictably random sequence, an iv for the sequence to start differently on each run)
3. Encryption. $C = M \oplus S$
⊕ Speed, Low Error propagation(errors in one bit do not affect subsequent symbols)
⊖ Low Diffusion (a change in a bit only affects one bit), Susceptibility to Modification (low diffusion makes it easier to tamper)
⊖ key stream generators are periodic because seed is finite. Thus, we need period long enough to not an issue (avoiding frequency analysis)

Linear Feedback Shift Register for Key Stream Generators
build a """random""" sequence of bits using a linear recurrence relation on a sequence of bit. For example: Starting with a state a_0, a_1, a_2, a_3 $a_n = a_{n-3} \oplus a_{n-4}$
⊕ Easy to build, and to analyze.

Randomness of LFSR
- if characteristic polynomial of the recurrence relation of the LFSR is primitive. The maximum possible number of states before repeating - For an L-bit register, the maximum period is $2^L - 1$ states
- **distribution property.** As a consequence, the sequence generated exhibit good distribution properties. Every possible non-zero state appears exactly once in a cycle.
However, the underlying operation is **linear**. (some algorithms can recover this with just a subsequence of the stream)