Distributed computer security

Computer security:

1. Secrecy (privacy, confidentiality)
2. Integrity
3. Availability (without *denial of service*)

Fault tolerance:

1. Reliability
2. Safety

Fault-tolerant and secure computer and communication system is called **dependable**.

Distributed systems are inherently more vulnerable to security threats than a single computer system:

- Open architecture
- Need for interaction across a wide range of autonomous and heterogeneous systems
- Message passing IPC through a communication network (*spoofing* and *forging*)

Fundamentals of computer security

Two views of computer security:

- **access control policy**: security policy describing how objects are to be accessed by subjects
- **flow control policy**: security policy describing the information flow between entities (objects and subjects)

Four categories of common security threats to objects:

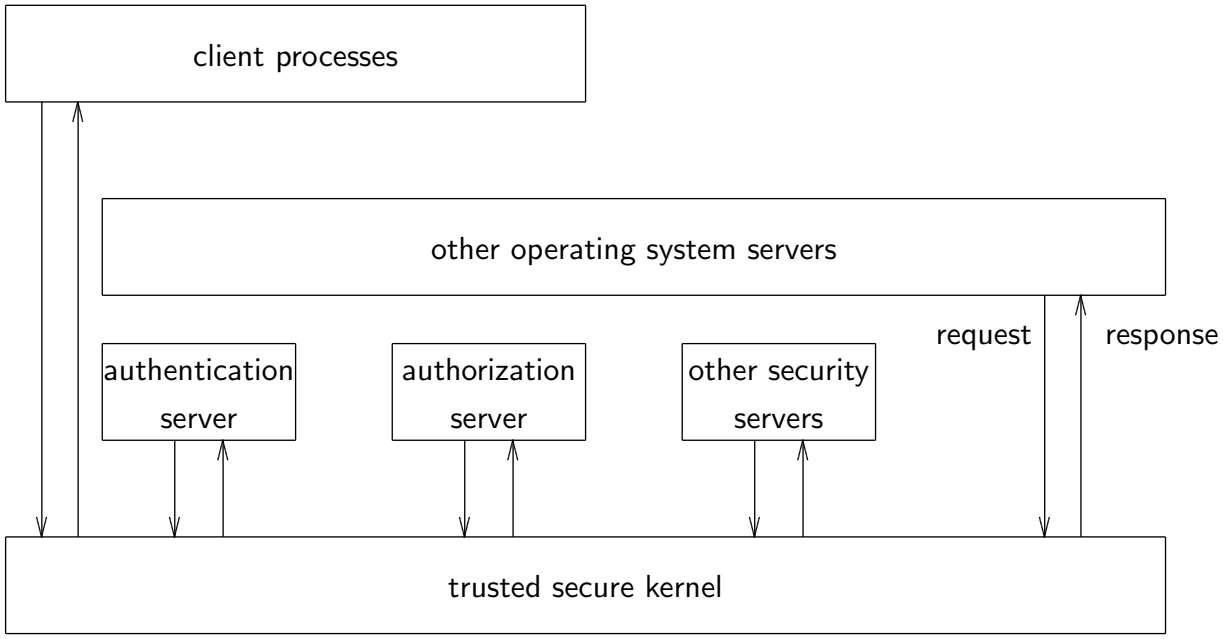
- interruption
- interception
- modification
- fabrication

Fundamental approaches in dealing with security problems:

- authentication (excluding external intruders)
- authorization (control of internal intruders)
- fault-tolerance (prevention of unintentional faults)
- encryption (maintaining privacy)
- auditing (passive form of protection, catching security breaches)

Security issues in distributed systems

Distributed OS system architecture principle: separation of mechanisms (kernel) and policies (servers).



Retaining interoperability and transparency in face of potential security threats - *security transparency*. To achieve this, a standard security system architecture with an API for trusted applications is needed. Example: Generic Security Service Application Program Interface (GSS-API).

Discretionary access control models

Provide access control on an individual basis.

- **Access control matrix** (ACM) using the *distributed*

compartment model (logical grouping of collaborating subjects and objects across node boundaries)
  - access based on *distributed handles* - application oriented, independent of the underlying operating system
  - each distributed compartment has at least one member - owner (has maximum privileges)
  - may be hierarchically structured
- Implementation of ACM: *access control lists* (ACL), *capability list* (CL), *lock-key* (combination of ACL and CL)

Mandatory flow control models

Concerned about information flow control on a *systemwide* basis.

- Categorizes all system entities into *security classes*
- Classification is labeled on every subject and object
- Access is controlled according to this classification
- The class of an entity seldom changes after it has been created

Lattice model

The best-known information flow model. A lattice is a directed acyclic graph (DAG) with a single source and sink. Each object and subject is associated with a security class, and all

security classes form a partially ordered set. Information can flow only in the direction that matches the partial ordering.

Formal definition of the lattice model:

$$FM =< S, O, SC, F, \oplus, \otimes, \rightarrow >$$

where

- S - set of subject (active agents responsible for information flow)
- O - set of objects (logical or physical information resources)
- SC - finite set of security classes corresponding to disjoint classes of information (all form a partial ordering)
- F - mapping function from S or O to SC called *binding*
- $\oplus$ - least upper bound operator on SC; for any two classes A and B, the class $A \oplus B$ is uniquely defined
- $\otimes$ - greatest lower bound operator on SC; for any two classes A and B, the class $A \otimes B$ is uniquely defined
- $\rightarrow$ - flow relation defined on pairs of security classes; $A \rightarrow B$ means information in class A is permitted to flow into class B (exists only of B is higher than A in partial ordering)

An FM is secure only if the execution of a sequence of operations cannot give rise to an information flow that violates the relation $\rightarrow$.

Properties of a lattice:

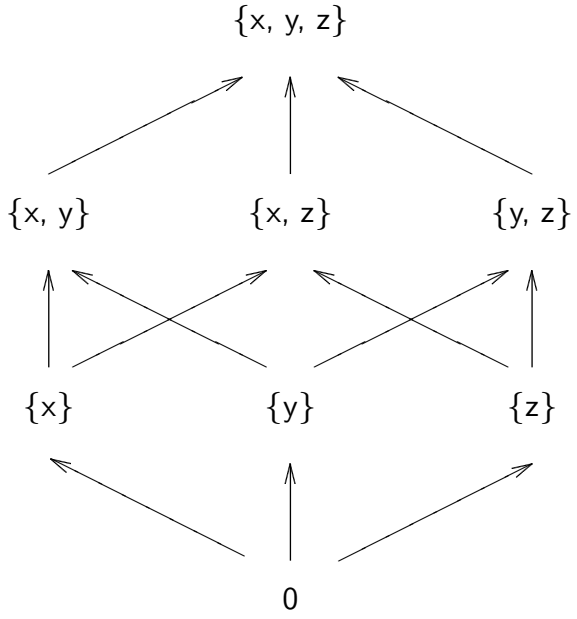
- Reflexive: $A \rightarrow A$
- Transitive: $A \rightarrow B$ and $B \rightarrow C$ implies $A \rightarrow C$
- Antisymmetric: $A \rightarrow B$ and $B \rightarrow A$ implies $A = B$
- Aggregation: $A \rightarrow C$ and $B \rightarrow C$ implies $A \cup B \rightarrow C$
- Separation: $A \cup B \rightarrow C$ implies $A \rightarrow C$ and $B \rightarrow C$

Examples of a lattice

A linear ordered lattice, in which $SC = C_1, \ldots, C_n$, $C_i \rightarrow C_j$ iff $i \leq j$, $C_i \oplus C_j = C_{max(i,j)}$, and $C_i \otimes C_j = C_{min(i,j)}$:

$$C_1 \rightarrow C_2 \rightarrow \ldots \rightarrow C_{n-1} \rightarrow C_n$$

A lattice of subsets of $X = x, y, z$, in which $SC = powerset(X), C_i \rightarrow C_j$ iff $C_i \subseteq C_j, C_i \oplus C_j = C_i \cup C_j$, and $C_i \otimes C_j = C_i \cap C_j$:



Bell-LaPadula model: Cartesian product of the linear ordered lattice (hierarchical) - *security level* - and the subset lattice (non-hierarchical) - *security category*.

Cryptography

Identities of clients and servers are called **principals**. Authenticated principal: principal given a secret key (a unique attribute). Authenticated message: data unit that carries a *digital signature* so that the message cannot be forged or repudiated. Cryptography can be applied for authentication of principals and signing of messages in distributed systems.

Private-key cryptographic systems

Algorithm decomposed into two parts: a *function* (public) and a *key* (secret). A single secret key is used to maintain a secret conversation between principals (for both encryption and decryption) - *symmetric cryptography*. Examples: DES, IDEA, AES (Rijndael). Problem: key distribution, large number of keys.

Public-key cryptographic systems

Introduced by Diffie and Hellman. Each principal maintains a pair of encryption and decryption keys, K_e and K_d. The encryption algorithm E and K_e are known to public. The decryption algorithm D and K_d are secret information belonging to the principal - *asymmetric cryptography*. Examples: RSA, Diffie-Hellman. Property of RSA:

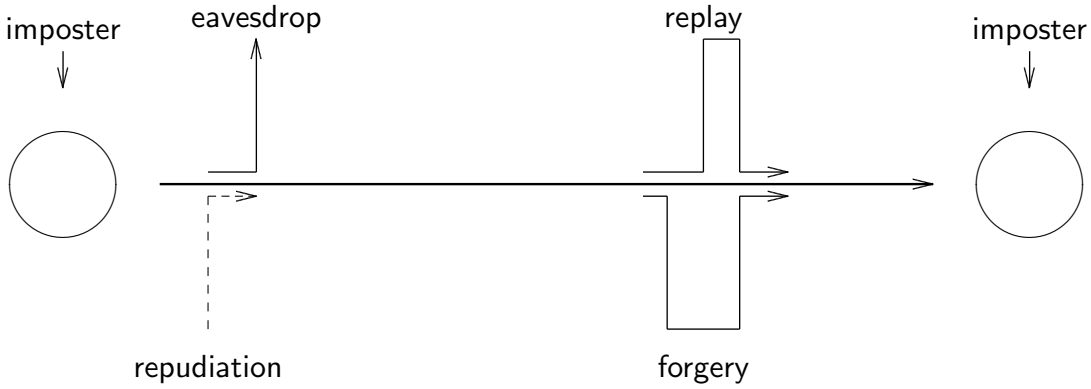
$$M = D_{K_s}(E_{K_p}(M)) = D_{K_p}(E_{K_s}(M))$$

where $K_s = K_d$ is the secret key, $K_p = K_e$ is the public key. The robustness of the algorithm is based on the computational complexity of factoring a large number upon which the keys are based.

Problems: higher computational complexity, public key distribution.

Authentication and key distribution

Authentication-related threats:



Final goal of the authentication protocol:

- For interactive connection-oriented services: to achieve a mutually trusted session key for the communicating processes
- For one-way connectionless services: authentication and protection of secrecy and integrity combined into a one-shot message

Most distributed applications follow the client/server programming paradigm; interaction is viewed as request/reply communication. Session keys can be used for this, but conceptually simpler is the notion of **tickets**.

The Kerberos protocol

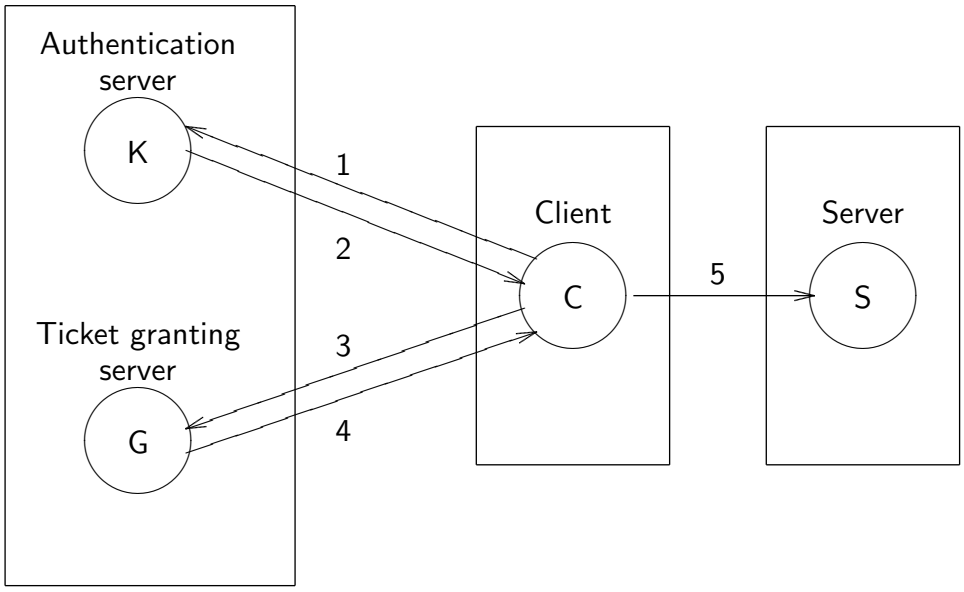
Designed for the client/server model

- Ticket: {identities, IP addresses, timestamp, lifetime, session key} encrypted with the server's key

- Authenticator: {identity, IP address, timestamp} encrypted with the session key

Authenticator and ticket pair - **credential**.

Kerberos 5 authentication protocol:



1. $C \rightarrow K : C, G, N$
2. $K \rightarrow C : \{K_{cg}, N\}_{K_C}, Ticket_{cg}$
3. $C \rightarrow G : Authenticator_{cg}, Ticket_{cg}$
4. $G \rightarrow C : \{K_{cs}, N\}_{K_{cg}}, Ticket_{cs}$
5. $C \rightarrow S : Authenticator_{cs}, Ticket_{cs}$