Kerberos

- Kerberos is a **network authentication protocol** that provides authentication for client-server applications
- An Authentication Server allows access
- **Ticket**: specifies that a particular client (authenticated by the Authentication Server) has the right to obtain service from a specified server S
- **Realm**: network under the control of an Authentication Server
Basic Authentication Protocol

C → AS: $\text{ID}_c \ || \ P_c \ || \ \text{ID}_s$

AS → C: Ticket

C → S: $\text{ID}_c \ || \ \text{Ticket}$

$\text{Ticket} = E_{K_s}[\text{ID}_c \ || \ P_c \ || \ \text{ID}_s]$  

- ID represents identifiers
- $P_C$ represents password of client
- E denotes encryption
- $K_s$ is a key shared by Authentication Server AS and server S
Improved Authentication Protocol

- Use two type of tickets with two different lifetimes:
  - One ticket grants to right to ask for service; performed once per login session $Ticket_{tgs}$
  - For each type of service, use a ticket that grants the right to use that particular service $Ticket_S$
  - Every time that service is needed, used the ticket $Ticket_S$

- Mark time when tickets are issued and also lifetime of tickets.
Overview of Kerberos

1. User logs on to workstation and requests service on host.

3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.

5. Workstation sends ticket and authenticator to server.

4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.

6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.
**Goal: Obtain Ticket-Granting Ticket**

\[ C \rightarrow AS: \ ID_c \ || \ ID_{tgs} \ || \ TS_1 \]

\[ AS \rightarrow C: \ E_{K_C} [K_{c,tgs} || ID_{tgs} || TS_2 || Lifetime_2 || Ticket_{tgs}] \]

\[ Ticket_{tgs} = E_{K_{tgs}} [K_{c,tgs} || ID_C || AD_C || ID_{tgs} || TS_2 || Lifetime_2] \]

- ID_{tgs} denotes the identifier of the Ticket Granting Server (TGS)
- TS1 and TS2 are timestamps
- K_C is the key shared by the AS and client C
- K_{C,tgs} is the key shared by the TGS and client C
- K_{tgs} key known by AS and the TGS
- Ticket_{tgs} is the ticket
- Lifetime is the validity of the ticket
- AD is address identifier
V4: Ticket-Granting Service Exchange

Goal: Obtain Service-Granting Ticket

\[ C \rightarrow \text{TGS: } ID_S \ || \ Ticket_{tgs} \ || \ Authenticator_C \]
\[ \text{TGS } \rightarrow \ C: \quad E_{K_{c,tgs}} [ K_{C,S} \ || \ ID_S \ || \ TS_4 \ || \ Ticket_S ] \]

\[ Ticket_{tgs} = E_{K_{tgs}} [ K_{C,tgs} \ || \ ID_C \ || \ AD_C \ || \ ID_{tgs} \ || \ TS_2 \ || \ Lifetime_2 ] \]
\[ Ticket_S = E_{K_S} [ K_{C,S} \ || \ ID_C \ || \ AD_C \ || \ ID_s \ || \ TS_4 \ || \ Lifetime_4 ] \]
\[ Authenticator_C = E_{K_{C,tgs}} [ ID_C \ || \ AD_C \ || \ TS_3 ] \]

\( K_S \) is the key shared by the TGS and server S
V4: Client-Server Authentication Exchange

**Goal: Obtain Service**

\[ C \rightarrow S: \quad \text{Ticket}_S \parallel \text{Authenticator}_C \]

\[ S \rightarrow C: \quad E_{KC,S} [ TS_5 + 1 ] \]

\[ \text{Ticket}_S = E_{KS} [ K_{C,S} \parallel ID_C \parallel AD_C \parallel ID_s \parallel TS_4 \parallel \text{Lifetime}_4 ] \]

\[ \text{Authenticator}_C = E_{KC,S} [ ID_C \parallel AD_C \parallel TS_5 ] \]
Request for Service in Another Realm

- Authenticate to local AS and obtain ticket to local TGS
- Ask local TGS for ticket for remote TGS, obtain ticket for remote TGS
- Ask remote TGS for ticket for remote server S, obtain ticket for remote server S
- Ask for service from remote server S
CS 526: Information Security
Access Control
Access Control

- Given a subject, what objects can it access and how?
- Given an object, what subjects can access it and how?
Definitions

- Restrictions on actions *after* authentication
- Subject = user, process, or thread (gets access)
- Object = resource (object is accessed)
  - In OS: file, directory, TCP/UDP port, shared memory segment, IO device, etc
  - In database system: table, view, procedure, etc
- Rights = different types of access
  - read, write, append, execute, own, etc
Goals

- Goals of access control include
  - Preventing access
  - Limiting access
  - Granting access
  - Revoking access

- Given a subject, what objects can it access? How?
- Given an object, what subjects can access it? How?

- The assignment of access rights is based on principle of least privilege (= access to only what is necessary)
A way to force Alice and Bob to modify Data_2 only by executing Update_Program

- Update_Program restricts the way updates to Data_2 are done (e.g., enforces checks that prevent corruption of Data_2 by Alice or Bob)

Note that Carol can replace Update_Program
Access Control Matrix Model

All subjects appear as rows, all objects as columns

```
<table>
<thead>
<tr>
<th>Subjects</th>
<th>Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- U has `rw own` access to F.
- U has `r` access to G.
- V has `rw own` access to F.
- V has `r` access to G.
Implementations of Access Matrix Model

- How to model changes?
- What if we need temporary rights?
- Access Matrix Model does not model the rules by which permissions can change in any particular system
- Access Matrix implemented as:
  - Access Control Lists
  - Capabilities
Storage of Access Control Matrix

- **Alternative 1**: A list for every row
  - Every subject has a Capability List (C-list)
  - Similar to adjacency list representation of the (directed) graph described by the matrix

- **Alternative 2**: A list for every column
  - Every object has an Access Control List (ACL)
  - Similar to adjacency list representation of the reverse of the graph described by the matrix
ACLs and C-lists for the above matrix

ACLs:

<table>
<thead>
<tr>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>U:r</td>
<td>U:r</td>
</tr>
<tr>
<td>U:w</td>
<td>V:r</td>
</tr>
<tr>
<td>U:own</td>
<td>V:w</td>
</tr>
</tbody>
</table>

C-lists:

<table>
<thead>
<tr>
<th>U</th>
<th>U:F,r, F:w, F:own, G:r</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>V:G:r, G:w, G:own</td>
</tr>
</tbody>
</table>
## Access Control Triplets

A commonly used approach in relational database systems involves the concept of access control triplets, which consist of a subject, an access type, and an object. Here is a table illustrating how these components are typically organized:

<table>
<thead>
<tr>
<th>Subject</th>
<th>Access</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>r</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td>w</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td>own</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td>r</td>
<td>G</td>
</tr>
<tr>
<td>V</td>
<td>r</td>
<td>G</td>
</tr>
<tr>
<td>V</td>
<td>w</td>
<td>G</td>
</tr>
<tr>
<td>V</td>
<td>own</td>
<td>G</td>
</tr>
</tbody>
</table>

This table reflects the common usage of access control triplets in relational database systems, where the subjects (U, V) can perform operations (r, w, own) on objects (F, G).
More on capabilities

• A protected reference to an object is a capability
  ◦ its possession by a subject suffices for access

• For example, a path (e.g., /etc/passwd) is not a capability, but a file descriptor is a capability
  ◦ because the table of file descriptors is in kernel memory, beyond the reach of user programs

• Easy to delegate (unlike ACL)
**ACLs vs C-lists**

- **ACLs**
  - easier to change rights to a particular object
  - easier to add or delete objects

- **C-lists**
  - easier to add or delete subjects
  - easier to delegate rights (simply pass along the corresponding capabilities)

- **ACLs are more commonly used than C-lists**
Confused Deputy Problem

- Compiler can write to any file (including File1)
- Alice has no access rights for File1
- Alice can invoke the compiler, and provide a filename where debugging information will be written
- What happens when Alice invokes the compiler and provides File1 as the debug filename?
Confused Deputy Problem (cont’d)

- Compiler is acting on behalf of (as “deputy” of) two masters
  - the invoker (Alice, who wants to compile a program)
  - the system administrator (who installed the compiler and controls File1 and other info)
- Which master is the deputy serving when performing a write?
- “Confused Deputy Problem”: When the result of Alice’s command is the overwriting of File1
- Correct behavior would be for Alice’s command to fail, because she has no right to access File1
Confused Deputy Problem (cont’d)

- Problem is that, when it is acting on Alice’s behalf, the compiler was using its own access rights rather than Alice’s.
- Compiler should use Alice’s access rights when running on Alice’s behalf.
- How does compiler learn Alice’s access rights?
  - Easiest is for Alice to delegate them to compiler.
  - If C-lists are used, Alice can pass her capabilities along to the compiler (simpler than with ACLs).
Who Decides Access Matrix Entries?

- **Discretionary Access Control (DAC):** Access rights are set at subject’s discretion
  - Notion of owner of an object
  - Used in UNIX, Windows, etc

- **Mandatory Access Control (MAC):**
  - Centrally controlled to enforce a security policy

- **MAC + DAC**
  - Mandatory takes precedence (i.e., subjects cannot override the security policy)
  - Discretionary can be used to further restrict (but not to relax) access
Unconstrained DAC Drawbacks

- Subjects can allow information flows that are contrary to the organization’s policy
  - Subjects may not do it on purpose, e.g., they run infected or buggy software they assumed was OK
- Invoked program may be doing another entity’s bidding, e.g.,
  - Program-provider if it put malware in its program
  - Input-provider if program is vulnerable to input-injected code
- Viruses & trojan horses can flourish
Unconstrained DAC Drawbacks (cont’d)

- Implicit assumptions, made by subjects, can cause harm when they do not hold
  - about software being correct and benign
  - about the behavior of others
  - about organization’s policy (misunderstanding it)

- Makes Confused Deputy Problem more likely

- Ill-suited for many policies
  - Multi-level, role-based, attribute-based, etc
Mandatory Access Control

- Mandatory access control (MAC) restricts the access of subjects to objects based on a system-wide policy.
- The system security policy (as set by the administrator) entirely determines the access rights granted:
  - denying users full control over the access to resources that they create.
The Need for MAC

- Host compromise by network-based attacks is the root cause of many serious security problems
  - Worm, Botnet, DDoS, Phishing, Spamming

- Why hosts can be easily compromised
  - Programs contain exploitable bugs
  - The discretionary access control mechanism in the operating systems was not designed to take buggy software in mind
Examples of MAC for Linux/Unix

- MAC: a system-wide security policy restricts the access rights of subjects
- Existing MACs for Linux / Unix:
  - SELinux, from NSA
  - AppArmor (SubDomain), from Novell Inc.
  - Systrace, from University of Michigan
  - LOMAC, from NAI Labs
  - …
Role-Based Access Control (RBAC)

- ACLs do not distinguish between different types of users
- RBAC assigns permissions to specific groups with meaning in the organization, rather than to low level data objects
- Makes administering security easier
RBAC

- Roles can be associated with other functional parameters
  - Only sysadmins between 9am-5pm
  - Only DB users at IP addresses inside the US

- Very complex roles can be developed

- Users may have multiple roles, but only one at a time.
Multi-level Security (MLS) Models

- Models are about “what” rather than “how”
  - Do not get into details of implementation
- In MLS models, objects and subjects have security levels
- Typical military security levels:
  - top secret \(\geq\) secret \(\geq\) confidential \(\geq\) restricted \(\geq\) unclassified
- Typical commercial security levels:
  - restricted \(\geq\) proprietary \(\geq\) sensitive \(\geq\) public
Bell-La Padula Model (BLP)

- A MAC Model for Multi-level Security
  - Concerns itself with *confidentiality* (preventing unwanted information flows)
- Security level of $X$ is denoted by $L(X)$
- Terminology
  - $L(S)$ for subject $S$ is called its security *clearance* (e.g., “Alice has a top-secret clearance”)
  - $L(O)$ for object $O$ is its security *classification* (e.g., “this file is classified as confidential”)
BLP requirements

- **Simple security rule**
  - Subject S can read Object O iff $L(S) \geq L(O)$
  - In words: “No read up”

- **Star property**
  - Subject S can write Object O iff $L(S) \leq L(O)$
  - In words: “No write down”
Biba Model

- MAC Model for Multi-level Security
  - Protection of *integrity*
  - Limiting the damage to integrity from corrupted data and from misbehaving subjects
- Integrity level of \( X \) is denoted by \( I(X) \)
  - Subject integrity level could reflect degree of confidence in (e.g.) software executing correctly
  - Object integrity level could reflect degree of confidence in (e.g.) data quality
Biba Model

- “No write up”
  - S can write O iff $I(S) \geq I(O)$
  - Avoids “contamination” of O by a lower-integrity S
  - Limits the damage that a compromised S can do

- “No read down”
  - S can read O iff $I(S) \leq I(O)$
  - Avoids “contamination” of S by a lower-integrity O
  - Limits the damage from a corrupted O
Extensions to Biba

- **Subject Low Water Mark**
  - “No write up”: S can write O iff \( I(S) \geq I(O) \)
  - S can always read O but, after reading,
    \[ I(S) = \min\{ I(S), I(O) \} \]

- **Object Low Water Mark**
  - S can read O iff \( I(S) \leq I(O) \)
  - S can always write O, but after writing
    \[ I(O) = \min\{ I(S), I(O) \} \]
Wall Security Policy

- **Goal:** Prevent information flows that lead to conflicts of interest
- **Data organized in conflict-of-interest classes**
- **Example 1:** Firm has clients A, B, and C who are competitors to each other
  - A, B, C form a conflict class
  - No employee of firm can be assigned to more than one of \{A,B,C\}
  - Employee with access to info about one of \{A,B,C\} must not have access to info about the other two
Example 2: Firm has two departments that engage in incompatible activities

- Department X deals with private ("insider") information not to be publicly disclosed (e.g., an investment bank advising corporate clients on mergers, acquisitions, initial public offerings)
- Department Y makes decisions based on public information (e.g., securities trading, managing mutual funds, etc)
- Need for information barrier between the two departments
Need-to-know Principle

- Even if someone has the necessary clearance level (say, top-secret) to access certain information, they should not necessarily be given access to all such information unless they have a need to know: that is, unless access to the specific information is necessary for the conduct of their duties in the organization.
- Need-to-know necessitates imposing further restrictions within a clearance level.
- Restrictions are done using DAC and/or categories.
Multilateral Security: Categories

- Categories are also known as compartments
- Examples of commercial security categories
  - Sales, R&D, HR, marketing, operations, IT
- Examples of military security categories
  - army, navy, air force, marines
  - nato, anzus
  - Alice might have a “top-secret” clearance that applies only within the “navy” category
Security Labels

- A label is a pair \((e,C)\) where \(e\) is a level, and \(C\) is a subset of categories
  - Labels = Levels \(\times\) (Powerset of Categories)
- Define an ordering relationship among labels
  - \((e_1, C_1) \leq (e_2, C_2)\) iff \(e_1 \leq e_2\) and \(C_1 \subseteq C_2\)
- This ordering relation is a partial order
  - reflexive, transitive, anti-symmetric
  - e.g., \(\subseteq\)
- Security labels form a lattice
Math Review: Partially Ordered Set

- A Set $S$ with relation $\leq$, written $(S, \leq)$, is called a partially ordered set if $\leq$ is
  - Anti-symmetric
    - If $a \leq b$ and $b \leq a$ then $a = b$
  - Reflexive
    - For all $a$ in $S$, $a \leq a$
  - Transitive
    - For all $a, b, c$, $a \leq b$ and $b \leq c$ implies $a \leq c$
An Example Security Lattice

- levels={top secret, secret}
- categories={army, navy}
Role-Based Access Control (RBAC)

- In RBAC, staff and users are not directly assigned access rights.
- Access rights are assigned to job functions in the organization (called roles).
- Staff and users are assigned roles:
  - They acquire access rights through these roles.
- Roles are often arranged as a hierarchy:
  - Similar to enterprise’s organizational chart.
  - Inheritance (ancestor subsumes descendants).
RBAC (cont’d)

- Greatly simplifies the management of access rights
- Used by most enterprises
- Widely known to be a “best practice”
- Can implement DAC, MAC, multilateral
- Role hierarchy can be any partial order
  - A tree in most commercial products
Attribute-Based Access Control (ABAC)

- Evolved from RBAC
- Roles + other attributes (of both users and resources)
- Examples of other attributes considered
  - Citizenship
  - Clearance level
  - IP address
  - Physical location (GPS coordinates)
  - Time
Covert Channels

- Covert channel of information flow = a communication channel based on the use of system resources not normally intended for communication
  - E.g., timing of packets sent, existence/attributes of file, lock/unlock file, timing/sizes of print jobs …
- Allow subjects to circumvent security policy
  - MLS systems are designed to restrict legitimate (overt) channels of communication (read/write)
Types of Covert Channels

- **Storage channel**
  - Uses attribute of shared resource

- **Timing channel**
  - Uses temporal/ordering relationship of access to shared resource

- **Noiseless channel**
  - Uses a resource available only to sender and receiver

- **Noisy channel**
  - Uses a resource available to subjects other than the sender and receiver, as well as to sender and receiver
Dealing with Covert Channels

- Impossible to eliminate all channels, but:
  - Covert channel attacks are hard to carry out
  - Their resulting bandwidth is typically very low