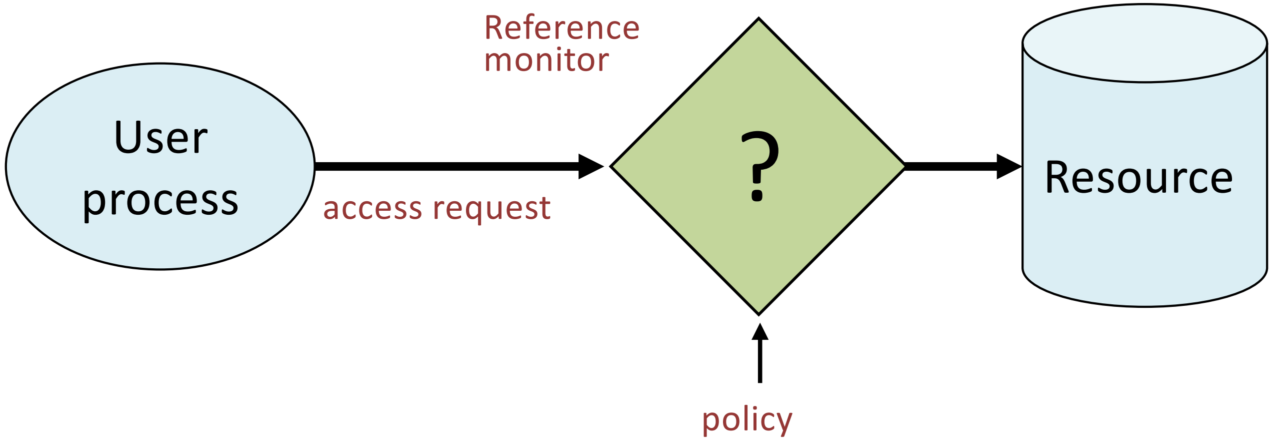
Logical Access Control and Security Architectures

Authorization, Access Control

- What is a subject/principal authorized to do?
- Definition of one or several perimeters of protection
  - Authorisations over assets/resources or groups of resources called “objects”
  - The authors of actions are called “subjects”
  - The “rights” granted to subjects over objects are formalized in a matrix
  - Security policy focused on the notion of perimeter
- The enforcement of these perimeters can intervene at different levels:
  - Applications
  - Operating system (or virtual machine)
  - Network

(Logical) Access control models

- Assumptions
  - System knows who the user is
    - Authentication via name and password, other credential
  - Access requests pass through gatekeeper (reference monitor)
    - System must not allow monitor to be bypassed



Access control matrix [Lampson]

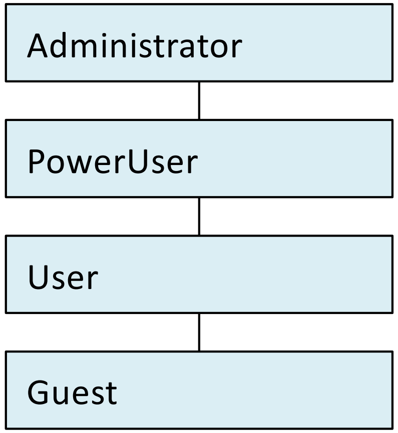
Objects

	File 1	File 2	File 3	…	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
…					
User m	read	write	read	write	read

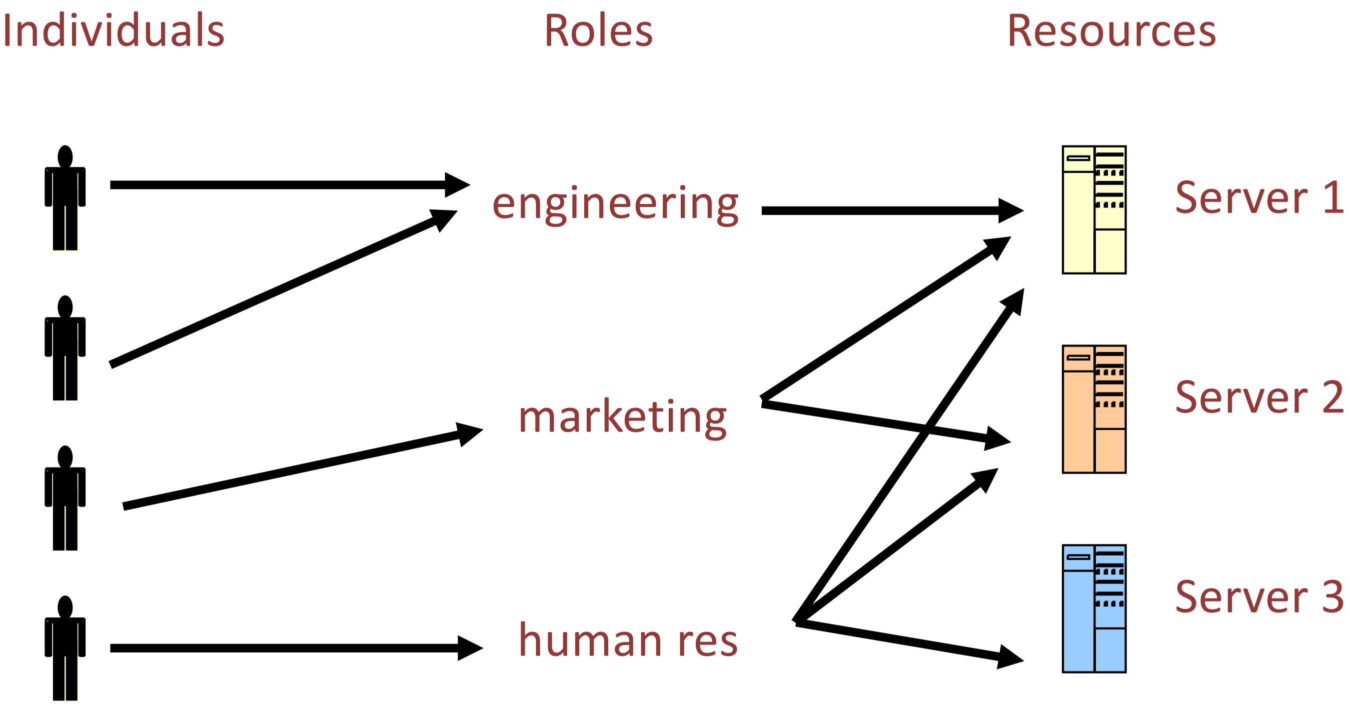
Subjects

Roles (aka Groups)

- Role = set of users
  - Administrator, PowerUser, User, Guest
  - Assign permissions to roles; each user gets permission
- Role hierarchy
  - Partial order of roles
  - Each role gets permissions of roles below
  - List only new permissions given to each role



Role-Based Access Control



Advantage: users change more frequently than roles

Implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - User holds a “ticket” for each resource
  - Two variations
    - store row of matrix with user, under OS control
    - unforgeable ticket in user space

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
User m	Read	write	write

- Access control lists are widely used in centralized contexts, often with groups
- Some aspects of capability concept are also used in a decentralized context (e.g. X.509v3 access control certificates)

ACL vs Capabilities

- Access control list
  - Associate list with each object
  - Check user/group against list
  - Relies on authentication: need to know user
- Capabilities
  - Capability is unforgeable ticket
    - Random bit sequence, or managed by OS
    - Can be passed from one process to another
  - Reference monitor checks ticket
    - Does not need to know identify of user/process

ACL vs Capabilities

- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- Revocation
  - ACL: Remove user or group from list
    - Or prevent process from acting as owner
  - Cap: Try to get capability back from process?
    - Possible in some systems if appropriate bookkeeping
      - OS knows which data is capability
      - If capability is used for multiple resources, have to revoke all or none …
    - Indirection: capability points to pointer to resource
      - If $C \rightarrow P \rightarrow R$, then revoke capability C by setting P=0

Access control: summary

- Access control involves reference monitor
  - Check permissions: $\langle$user info, action$\rangle \rightarrow$ yes/no
  - Important: no way around this check
- Access control matrix
  - Access control lists vs capabilities
  - Advantages and disadvantages of each
- Role-based access control
  - Use group as “user info”; use group hierarchies

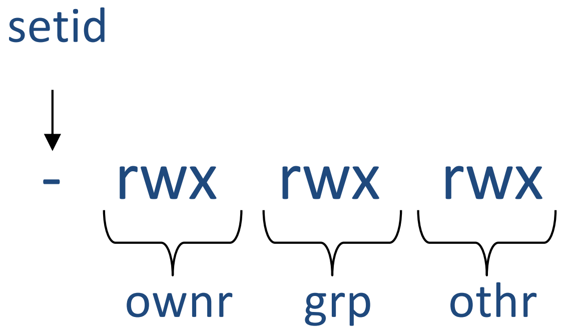
Unix access control

- Process has user id
  - Inherit from creating process
  - Process can change id
    - Restricted set of options
  - Special “root” id
    - All access allowed
- File has access control list (ACL)
  - Grants permission to user ids
  - Owner, group, other

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
User m	Read	write	write

Unix file access control list

- Each file has owner and group
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other
  - Represented by vector of four octal values
- Only owner, root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits – Discussed in a few slides



Principles of Secure Design

- Compartmentalization
  - Principle of least privilege
  - Isolation
- Defense in depth
  - Use more than one security mechanism
  - Secure the weakest link
  - Fail securely
- Keep it simple

Least Privilege Principle

- What’s a privilege?
  - Ability to access or modify a resource
- Assume compartmentalization and isolation
  - Separate the system into isolated compartments
  - Limit interaction between compartments
- Least Privilege?
  - A system module should only have the minimal privileges needed for its intended purposes

Isolation between processes

- Processes in OS:
  - A process may access files, network sockets, ….
    - Permission granted according to UID
  - Two processes with same UID have the same permissions
- Processes and privileges :
  - Compartment defined by UID (User ID)
  - Privileges defined by actions allowed on system resources

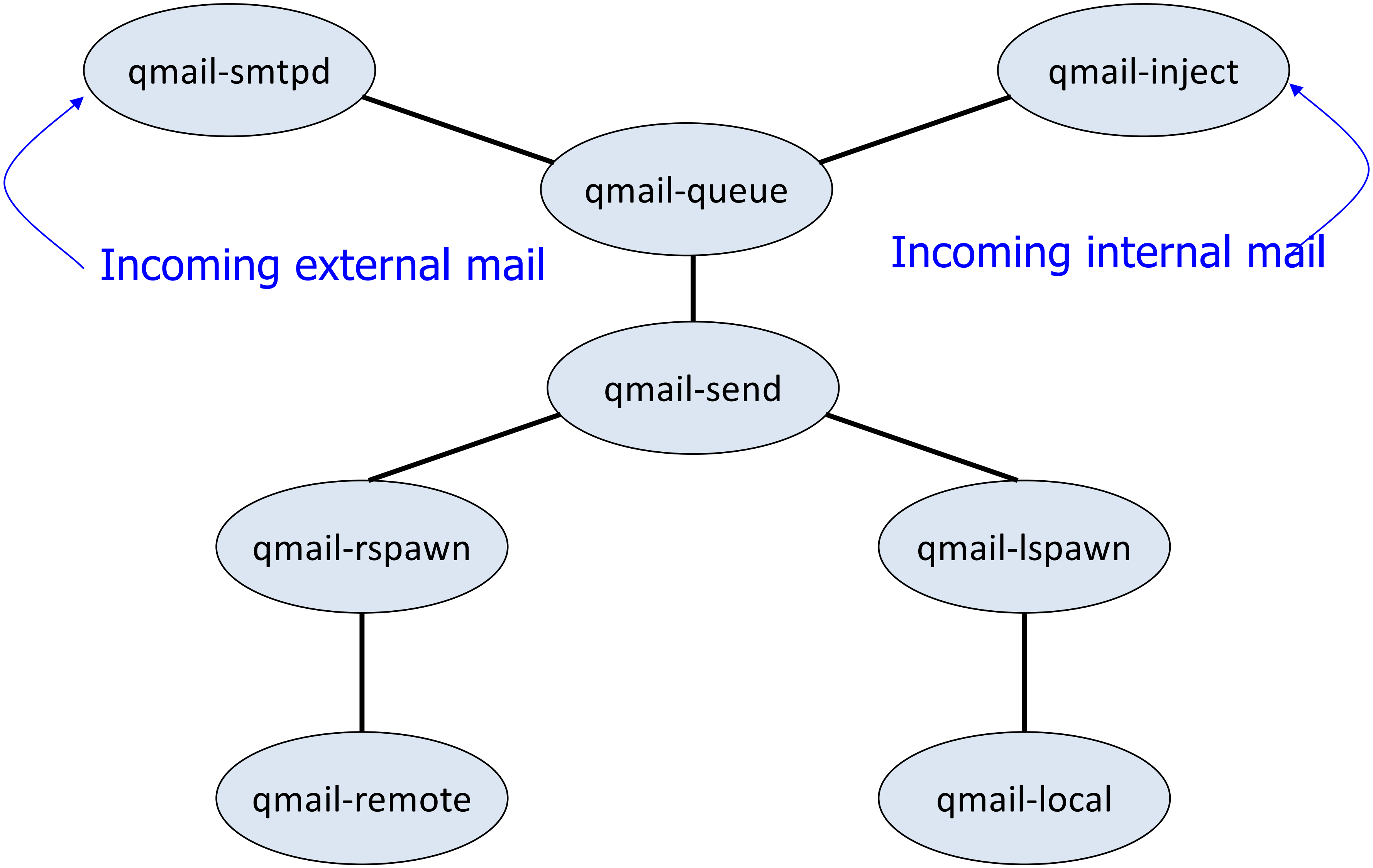
Example: Mail Agent

- Requirements
  - Receive and send email over external network
  - Place incoming email into local user inbox files
- Sendmail
  - Traditional Unix
  - Monolithic design
  - Historical source of many vulnerabilities
- Qmail
  - Compartmentalized design

Qmail design

- Mail Transfer Agent (MTA)
  - Sendmail replacement
- Function isolation based on OS isolation
  - Separate modules run as separate “users”
  - Each user only has access to specific resources
- Least privilege
  - Minimal privileges for each UID
  - Only one “setuid” program
    - setuid allows program to run as different users
  - Only one “root” program
    - root program has all privileges

Structure of qmail



Unix: Process effective user id (EUID)

- Each process has three Ids (+ more under Linux)
  - Real User ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
    - System call *access()* determines permission based on RUID
  - Effective User ID (EUID)
    - from set user ID bit on the file being executed, or sys call
    - determines the permissions for process
      - file access and port binding
  - Saved User ID (SUID)
    - So previous EUID can be restored
- Real Group ID, Effective Group ID, used similarly

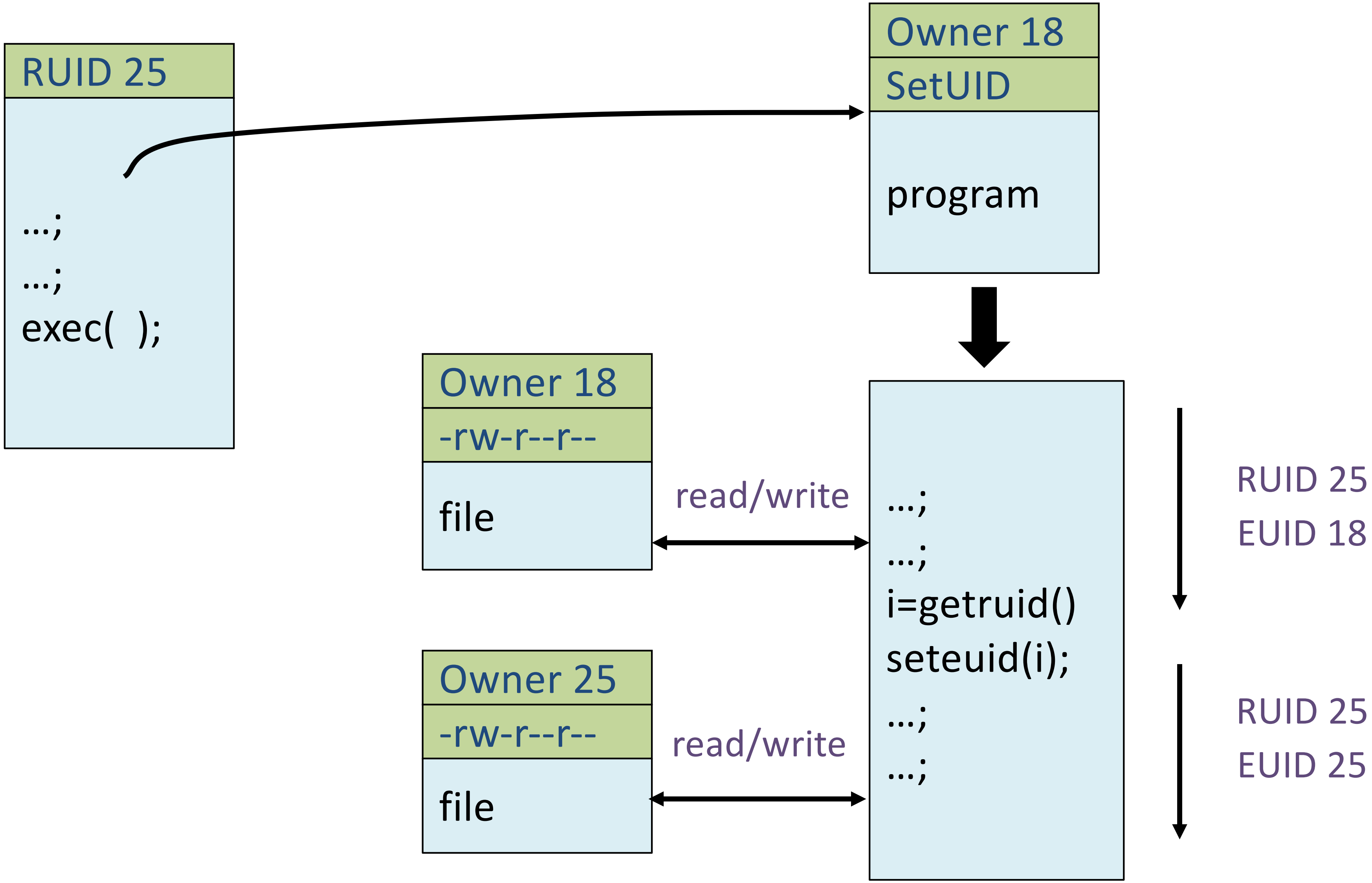
Unix: Setid bits on executable file

- Three setid bits
  - Setuid – set EUID of process to ID of file owner
    - “chmod u+s”
  - Setgid – set EGID of process to GID of file
    - “chmod g+s”
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory, but all owners with a write permission can modify it
    - ”chmod +t”

Unix: Process Operations and IDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs
  - except exec of file with setuid bit (then: euid receives uid of owner of process invoked)
- Setuid system calls
  - setuid(newid) can set RUID to newid (if you have the right to, e.g. root)
    - However user root or program setuid that changes to newid != 0 cannot regain root privileges !
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0
  - Details are actually more complicated: several different calls: setuid, seteuid, setreuid, setgid, …

Example


RUID 25
…;
…;
exec( );
Owner 18
SetUID
program
Owner 18
-rw-r--r--
file
read/write
Owner 25
-rw-r--r--
file
read/write
…;
…;
i=getruid()
seteuid(i);
…;
…;
RUID 25
EUID 18
RUID 25
EUID 25

Unix summary

- Advantages
  - Some protection from most users
  - Flexible enough to make things possible
- Main limitations
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

Mandatory Access Control (MAC)

- Access Control Models: Mandatory vs. Discretionary
  - Discretionary (DAC) = at the discretion of the resource owner
- A means of restricting access to objects based on the sensitivity of the information contained in the objects and whether they are authorized to access information of such sensitivity
- Authorization is based on prerequisites being met, resulting in an individual gaining access
- Enables the ability to deny users full control over the access to resources that they create
- access control is based on the compatibility of the security properties of the data and the clearance properties of the individual

SELinux

- Originally started by the Information Assurance Research Group of the NSA, working with Secure Computing Corporation.
- Based on a strong, flexible mandatory access control architecture based on Type Enforcement, a mechanism first developed for the LOCK system
- Originally started as two prototypes: DTMach and DTOS which were eventually transferred over to the Fluke research operating system
- Eventually the architecture was enhanced and renamed Flask. The NSA has now integrated the Flask architecture with Linux (SELinux)

SELinux: Background

- An example of how mandatory access controls can be added into Linux
  - Confining actions of a process, including a superuser process
- The security mechanisms implemented in the system provide flexible support for a wide range of security policies.
- Make it possible to configure the system to meet a wide range of security requirements.
- Documentation and source code is provided.

Authentication / Identification

Authentication

- What I know: passwords, secret questions, secret keys, tokens
- What I own: smartcard, bank card, RSA key, mobile phone
- What I am: biometrics – physiology (fingerprints, iris, veins, face, voice …) or behavior (signature, gestures …)

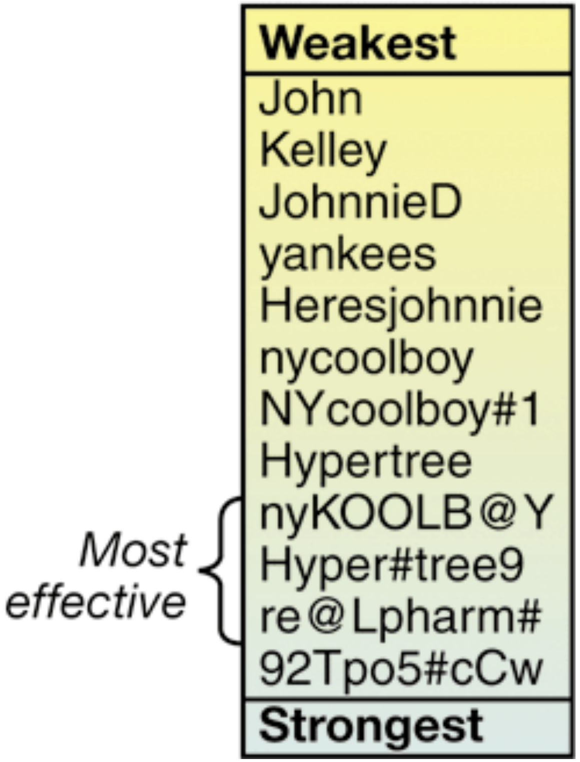
- Increasing association of two mechanims
  - two-factor authentication from different categories

Passwords

- Strictly personal
- Hard to find, easy to remember (no need to write it down)
  - Minimum number of characters (including ; / ! % ....)
  - Does not correspond to a dictionary word
- Avoid
  - Names or first names of relatives
  - Telephone number…
- Must be changed periodically
- Key phrase method
  - Saying, movie title….

FIGURE 11.15 • Passwords, weakest to strongest

Strong passwords use words that are unrelated to your interests and include upper- and lowercase letters, numbers, and symbols.



Weakest

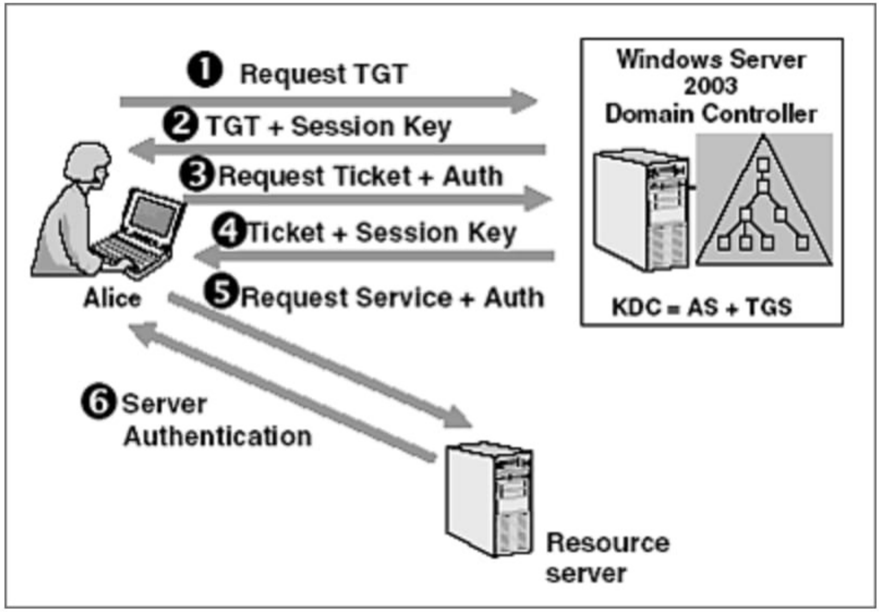
John
Kelley
JohnnieD
yankees
Heresjohnnie
nycoolboy
NYcoolboy#1
Hypertree
nyKOOLB@Y
Hyper#tree9
re@Lpharm#
92Tpo5#cCw

Strongest

Most effective

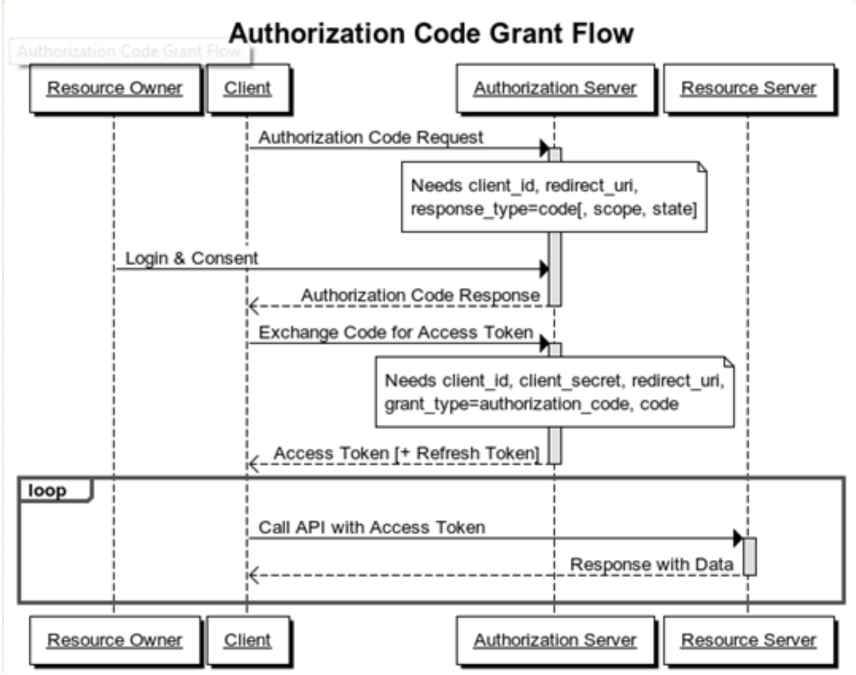
Token based authentication (and authorization)

- Kerberos
  - single domain sign-on (SDSSO) protocol
  - secret key cryptography
  - Client requests an authentication ticket (TGT) from the Key Distribution Center (KDC)
- OAuth2
  - Web authorization protocol
  - Delegation: authorise some third party to use or alter data that you own on a given web service without disclosing your credentials
  - Needs centralized server or database
- OpenID-Connect
  - multi domain single-sign-on protocol (MDSSO)
  - Built on top of OAuth2
- JWT (JSON Web Tokens)
  - Open standard for JSON-based tokens
  - Authentication and authorization: securely transmit information to unverified client
  - Lightweight and self-contained token, easy to generate and validate
  - does not require a centralized server or database
  - No revocation or refreshment of tokens



Token based authentication (and authorization)

- Kerberos
  - single domain sign-on (SDSSO) protocol
  - secret key cryptography
  - Client requests an authentication ticket (TGT) from the Key Distribution Center (KDC)
- OAuth2
  - Web authorization protocol
  - Delegation: authorise some third party to use or alter data that you own on a given web service without disclosing your credentials
  - Needs centralized server or database
- OpenID-Connect
  - multi domain single-sign-on protocol (MDSSO)
  - Built on top of OAuth2
- JWT (JSON Web Tokens)
  - Open standard for JSON-based tokens
  - Authentication and authorization: securely transmit information to unverified client
  - Lightweight and self-contained token, easy to generate and validate
  - does not require a centralized server or database
  - No revocation or refreshment of tokens

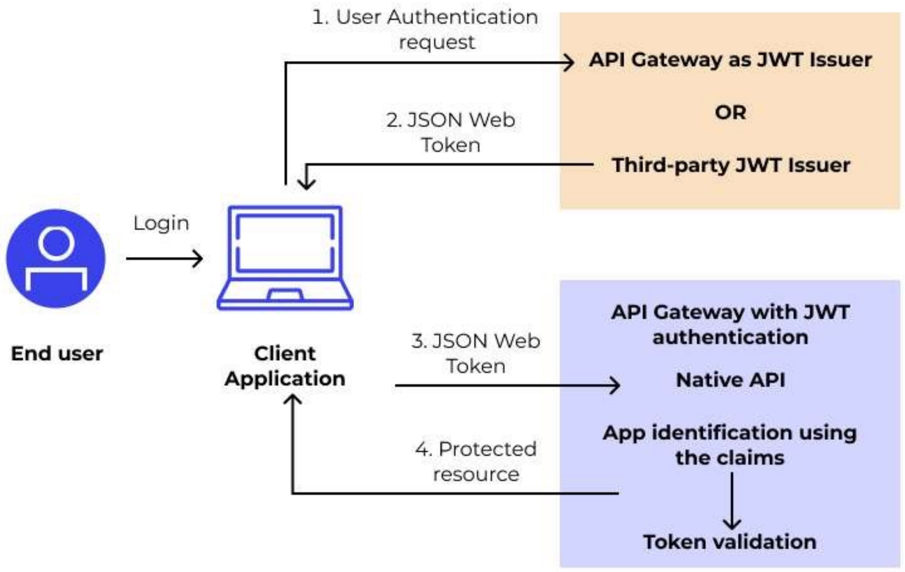

Authorization Code Grant Flow
Authorization Code Grant Flow
Resource Owner
Client
Authorization Server
Resource Server
Authorization Code Request
Needs client_id, redirect_uri, response_type=code[, scope, state]
Login & Consent
Authorization Code Response
Exchange Code for Access Token
Needs client_id, client_secret, redirect_uri, grant_type=authorization_code, code
Access Token [+ Refresh Token]
loop
Call API with Access Token
Response with Data
Resource Owner
Client
Authorization Server
Resource Server

OAuth2

Token based authentication (and authorization)

- Kerberos
  - single domain sign-on (SDSSO) protocol
  - secret key cryptography
  - Client requests an authentication ticket (TGT) from the Key Distribution Center (KDC)
- OAuth2
  - Web authorization protocol
  - Delegation: authorise some third party to use or alter data that you own on a given web service without disclosing your credentials
  - Needs centralized server or database
- OpenID-Connect
  - multi domain single-sign-on protocol (MDSSO)
  - Built on top of OAuth2
- JWT (JSON Web Tokens)
  - Open standard for JSON-based tokens
  - Authentication and authorization: securely transmit information to unverified client
  - Lightweight and self-contained token, easy to generate and validate
  - does not require a centralized server or database
  - No revocation or refreshment of tokens

JWT Authentication workflow

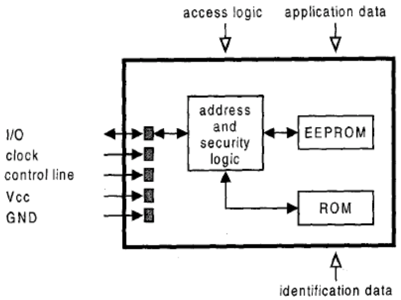


Smartcard

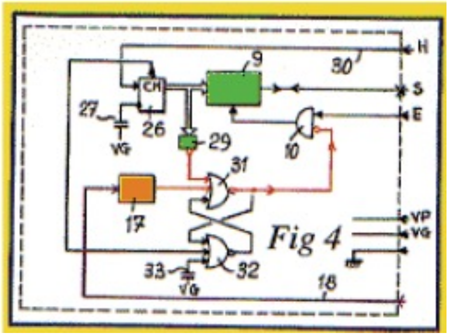
- Smart card, IC card, chip card, pin card …
- Main features:
  - Portable object storing data and/or procedures.
  - Secure object
  - Prevents reading data stored in the card memory (secret keys…)
  - Code executed in a trusted space
  - Low cost object customizable for hundreds of millions of users.
    - 1-5$ for SPOM / 0,1-0,5$ for magnetic cards
  - Can’t work alone and requires
    - A card reader to deliver energy
    - *A clock*
    - *A communication link*

The card and its technical features

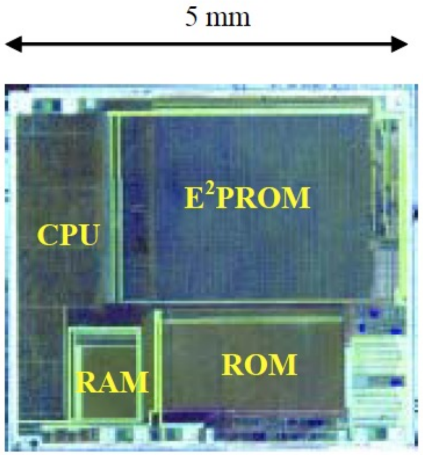
- Memory card:
  - Simple memory (reading / writing) (EPROM / EEPROM)
  - Not standardised



- Cabled logic Card :
  - The card contains a cabled device for protecting the data
  - Dedicated electronics to connect input and output pins
  - Not standardised



- Microprocessor card (SPOM/MAM) :
  - Memory + processor → programmable
  - Security algorithms (ex: DES, RSA)
  - ISO 7816 standards
  - Contact-based or contactless card
  - 1st implementation Bull CP8: 36b of RAM, 1 Kb EPROM, 1,6 Kb ROM



Biometrics

- Goals:
  - The user does not risk losing his authentication mechanism
  - Some authentification modes are well intégrated – ex: fingerprints (iPAQ, iPhone 5s)

Place Your Finger
Lift and rest the edge of your finger on the Home button repeatedly.

- Problems:
  - Biometric factor theft: never use 1-factor identification!
  - Sometimes easy counterfeating (fingerprints…)
  - The data should be preserved under the control of their owner (ex: smartcard)

FIGURE 11.17 • Facial recognition

By taking measurements of 128 facial features and matching them to the measurements of known faces, biometric software can help identify people.