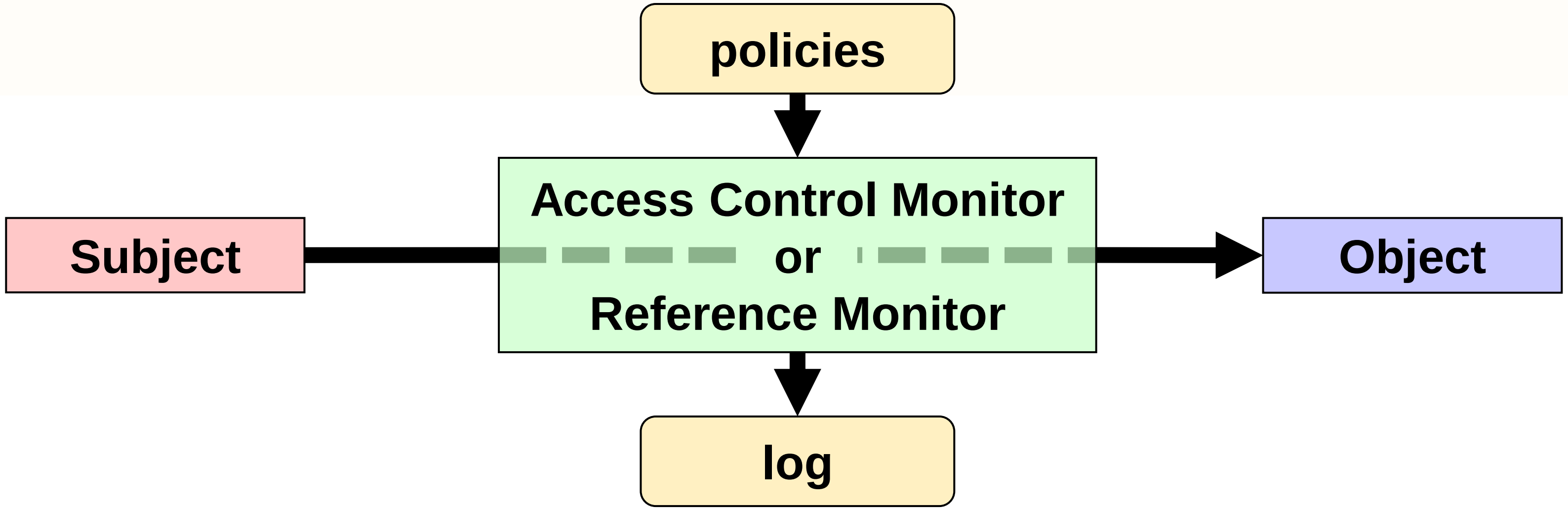
Access control models

Access types

- Physical access
  - Physical contact between a subject and the object of interest
    - Facility, room, network, computer, storage device, authentication token, etc.
  - **Out of scope of this course …**

- Informatic or electronic access
  - Information-oriented contact between a subject and the object of interest
    - Contact through request-response dialogs
  - Contact is mediated by
    - Computers and networks
    - Operating systems, applications, middleware, devices, etc.

Access control



- Definition
  - The policies and mechanisms that mediate the access of a subject to an object
- Normal requirements
  - Authentication
    - With some Level of Assurance (LoA)
  - Authorization
  - Accountability ➔logging

} AAA

Access control

- Subjects and objects
  - Both digital entities
  - Subjects can be something exhibiting activity :
    - Processes
    - Computers
    - Networks
  - Objects can be the target of an action :
    - Stored data
    - CPU time
    - Memory
    - Processes
    - Computers
    - Network
- An entity can be both subject and object

Least privilege principle

> Every program and every user of the system should operate using the least set of privileges necessary to complete the job
>
> J. H. Saltzer, M. D. Schroeder,
> The protection of information in computer systems, Proc. of the IEEE, 63(9) 1975

- Privilege:
  - Authorization to perform a given task
  - Similar to access control clearance
- Each subject should have, at any given time, the exact privileges required to the assigned tasks
  - Less privileges than the required create unsurpassable barriers
  - More privileges than the required create vulnerabilities
    - Damage resulting from accidents or errors
    - Potential interactions among privileged programs
    - Misuse of a privileges
    - Unwanted information flows
      - "need-to-know" military restrictions

Access control models

	O1	O2	…	Om-1	Om
S1		Access rights			
S2					
…					
Sn-1					
Sn					

- Access control matrix
  - Matrix with all access rights for subjects relatively to objects
  - Represents a conceptual model of the organization

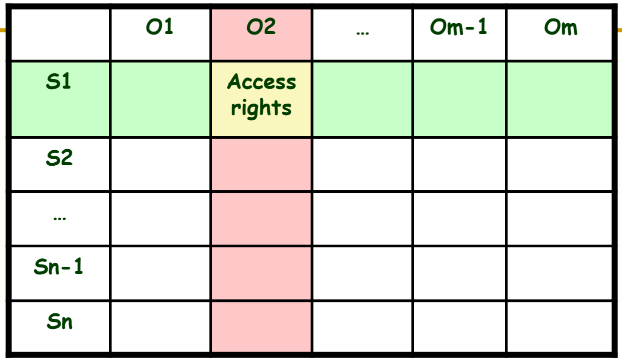
Access control models

	O1	O2	…	Om-1	Om
S1		Access rights			
S2					
…					
Sn-1					
Sn					

▷ ACL-based mechanisms

- ACL: Access Control List (matrix column)
  - List of access rights for specific subjects
  - Access rights can be positive or negative
  - Default subjects may often be used
- Usually ACLs are stored along with objects
  - e.g. for file system objects.
- Rights are then mapped to specific actions
  - Same right may map to different actions on different contexts

Access control models



	O1	O2	…	Om-1	Om
S1		Access rights			
S2					
…					
Sn-1					
Sn					

- Capability-based mechanisms
  - Capability: unforgeable authorization token (matrix row)
    - Contains object references and access rights
  - Access granting
    - Transmission of capabilities between subjects
  - Usually capabilities are kept by subjects
    - e.g. OAuth 2.0 access tokens

Access control kinds:
MAC and DAC

- Mandatory access control (MAC)
  - Access control policy statically implemented by the access control monitor
  - Access control rights cannot be tailored by subjects or object owners

- Discretionary access control (DAC)
  - Some subjects can update rights granted or denied to other subjects for a given object
    - Usually this is granted to object owners and system administrators

Access control kinds:
Role-Based Access Control (RBAC)

D.F. Ferraiolo and D.R. Kuhn, "Role Based Access Control“, 15th National Computer Security Conference, Baltimore, October 1992

- Not DAC or MAC
  - Roles are dynamically assigned to subjects
    - For access control it matters the role played by the subject and not the subject’s identity

- Access control binds roles to (meaningful) operations
  - Operations are complex, meaningful system transactions
    - Not the ordinary, low-level read/write/execute actions on individual objects
  - Operations can involve many individual lower-level objects

Access control kinds:
RBAC rules (1/2)

- Role assignment:
  - All subject activity on the system is conducted through transactions
    - And transactions are allowed to specific roles
    - Thus all active subjects are required to have some active role
  - A subject can execute a transaction **iff**
    - it has selected
  - or
    - been assigned
    - a role which can use the transaction

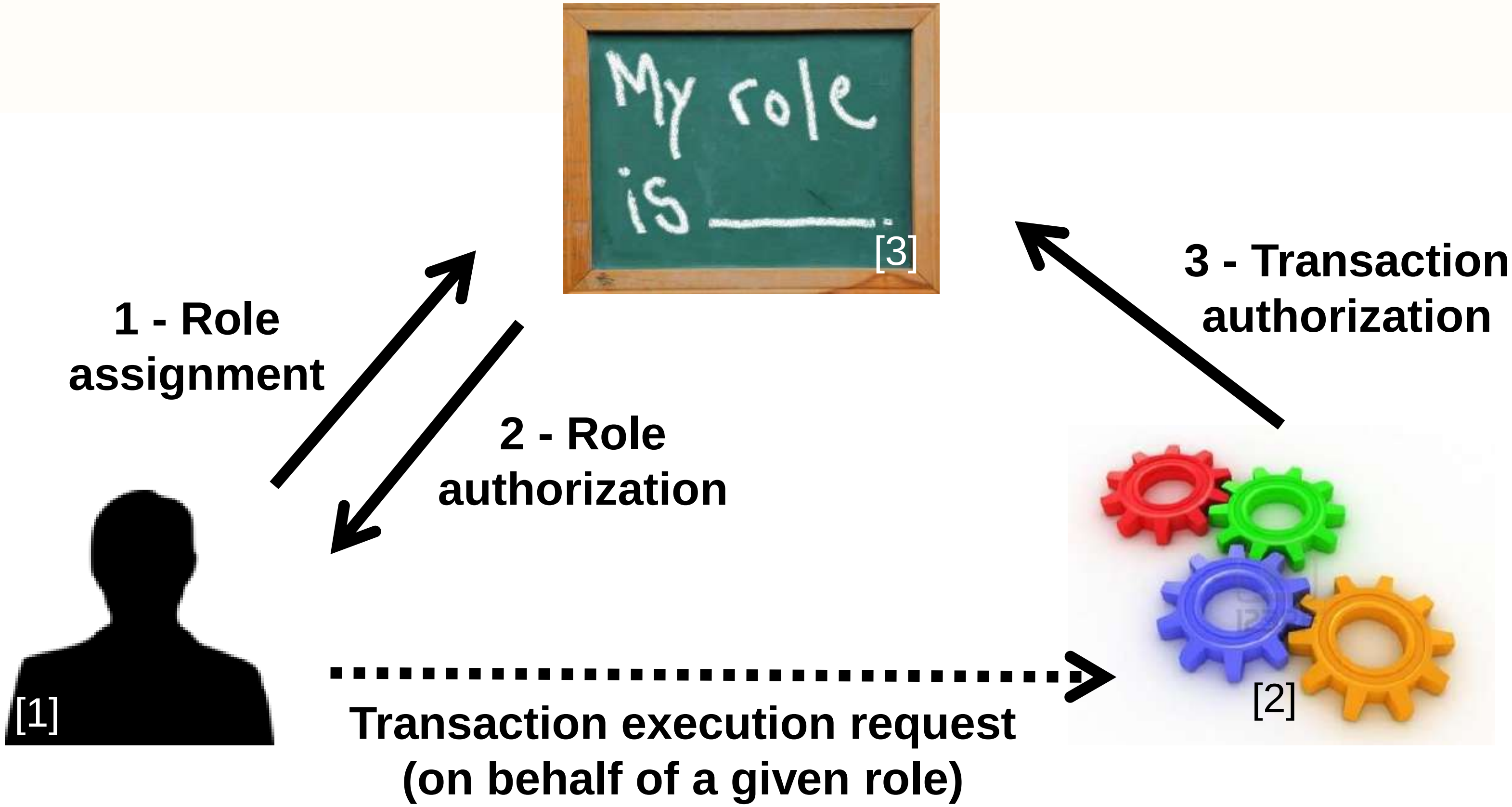
Access control kinds: RBAC rules (2/2)

- Role authorization:
  - A subject's active role must be authorized for the subject
- Transaction authorization:
  - A subject can execute a transaction **iff**
    - the transaction is authorized through the subject's role memberships

    and

    - there are no other constraints that may be applied across subjects, roles, and permissions

RBAC rules



1 - Role assignment

2 - Role authorization

3 - Transaction authorization

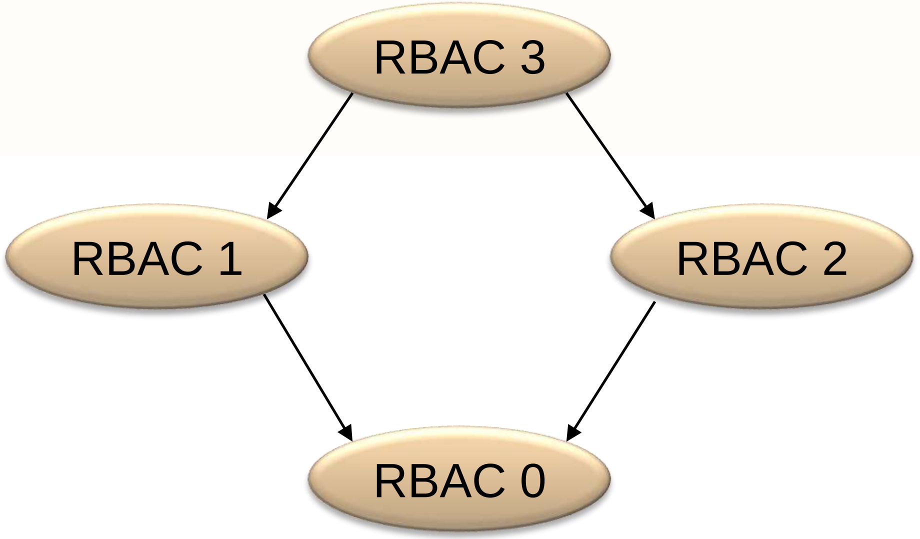
Transaction execution request (on behalf of a given role)

[1] From http://www.clker.com/clipart-24011.html
[2] From http://www.123rf.com/photo_12115593_three-dimensional-colored-toothed-wheels.html
[3] From http://www1.yorksolutions.net/Portals/115255/images/MyRoleIs.jpg

RBAC:
Roles vs. groups

- Roles are a collection of permissions
  - The permissions are granted to the subjects that, at a given instant, play the role
  - A subject can only play a role at a given time
- Groups are a collection of users
  - And permissions can be granted both to users and groups
  - A subject can belong to many groups at a given time
- The session concept
  - Role assignment is similar to a session activation
  - Group membership is ordinarily a static attribute

RBAC variants



- RBAC 0
  - No role hierarchies
  - No role constraints
- RBAC 1
  - RBAC 0 w/ role hierarchies (privilege inheritance)
- RBAC 2
  - RBAC 0 w/ role constraints (separation of duties)
- RBAC 3
  - RBAC 1 + RBAC 2

NIST RBAC model

- Flat RBAC
  - Simple RBAC model w/ user-role review
  - Role provides specific permissions for the user

- Hierarchical RBAC
  - Flat RBAC w/ role hierarchies (DAG or tree)
  - General and restricted hierarchies, where Roles gain additional permissions from other roles

- Constraint RBAC
  - RBAC w/ role constraints for separation of duty
  - Static: Conflicting Roles cannot be assigned
  - Dynamic: Subject cannot activate conflicting Roles within session

- Symmetric RBAC
  - RBAC w/ organization wide permission-role review
  - Allows review of a subject roles to prevent bloat

Access control kinds:
Context-Based Access Control (CBAC)

- Access rights have an historical context
  - The access rights cannot be determined without reasoning about past access operations
  - Example:
    - Stateful packet filter firewall
- Chinese Wall policy
  - Conflict groups
  - Access control policies need to address past accesses to objects in different members of conflict groups

D.F.C. Brewer and M.J. Nash, "The Chinese Wall Security Policy “, IEEE Symposium on Security and Privacy, 1989

Access control kinds: Attribute-Based Access Control (ABAC)

- Access control decisions are made based on attributes associated with relevant entities

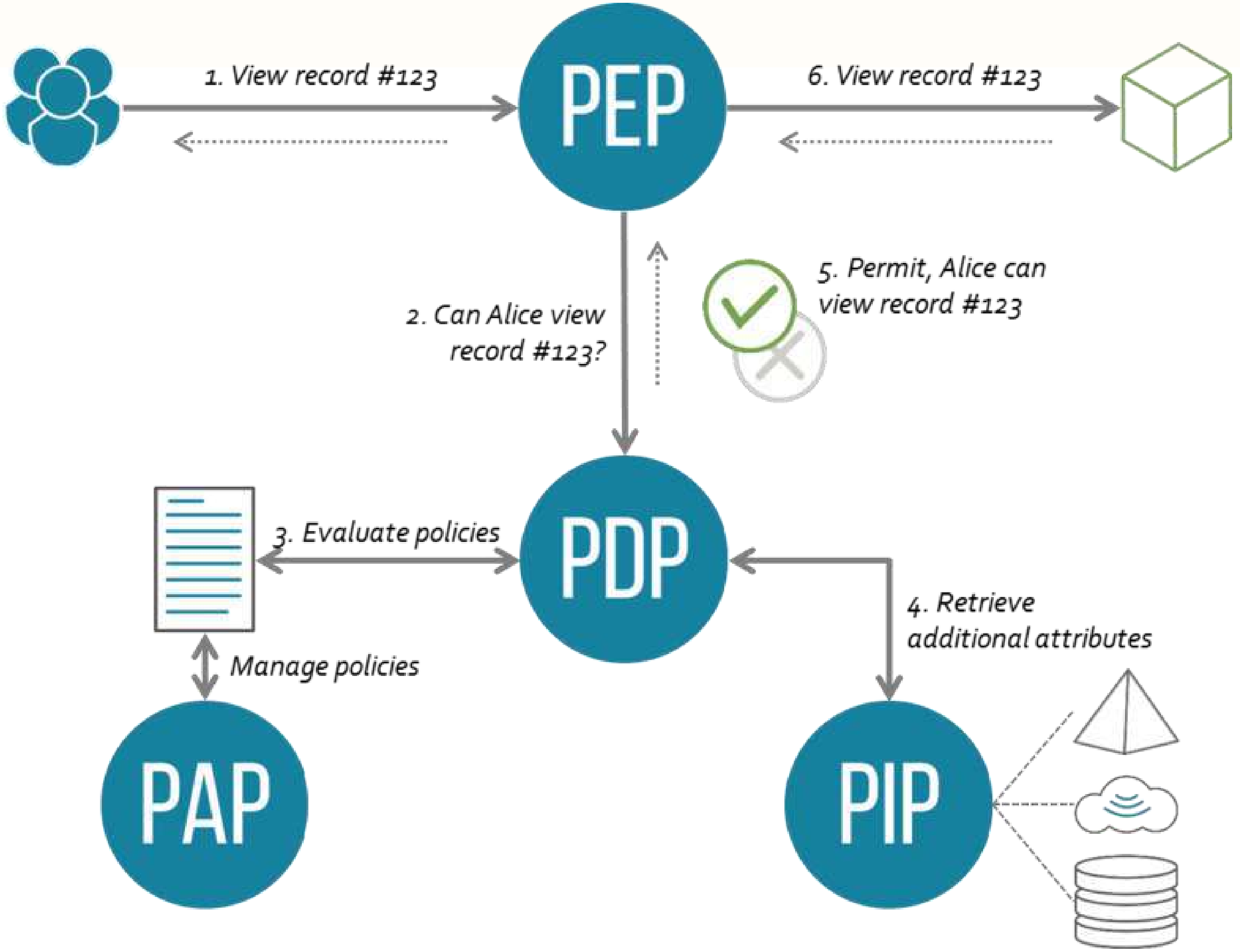
- OASIS XACML architecture
  - Policy Administration Point (PAP)
    - Where policies are managed
  - Policy Decision Point (PDP)
    - Where authorization decisions are evaluated and issued
  - Policy Enforcement Point (PEP)
    - Where access requests to a resource are intercepted and confronted with PDP’s decisions
  - Policy Information Point (PIP)
    - Provides external information to a PDP

XACML:
Access control with PEP and PDP

- A subject sends a request
  - Which is intercepted by the Policy Enforcement Point (PEP)
- The PEP sends the authorization request to the Policy Decision Point (PDP)
- The PDP evaluates the request against its policies and reaches a decision
  - Which is returned to the PEP

  - Policies are retrieved from a Policy Retrieval Point (PRP)
  - Useful attributes are fetched from Policy Information Points (PIP)
  - Policies are managed by the Policy Administration Point (PAP)

XACML big picture


1. View record #123
PEP
6. View record #123
2. Can Alice view record #123?
5. Permit, Alice can view record #123
3. Evaluate policies
PDP
4. Retrieve additional attributes
Manage policies
PAP
PIP

From https://en.wikipedia.org/wiki/XACML

Break-the-glass access control model

- It may be required to overcome the established access limitations
  - e.g. in a life threatening situation

- The subject may be presented with a break-the-glass decision upon a deny
  - Can overcome the deny at their own responsibility
  - Logging is fundamental to prevent abuses
    - Subject may have to justify action, after using the elevated right

Separation of duties

R.A. Botha, J.H.P. Eloff, “Separation of duties for access control enforcement in workflow environments”, IBM Systems Journal, 2001

- Fundamental security requirement for fraud and error prevention
  - Dissemination of tasks and associated privileges for a specific business process among multiple subjects
  - Often implemented with RBAC

- Damage control
  - Segregation of duties helps reducing the potential damage from the actions of one person
  - Some duties should not be combined into one position

Segregation of duties:

ISACA (Inf. Systems Audit and Control Ass.) matrix guideline

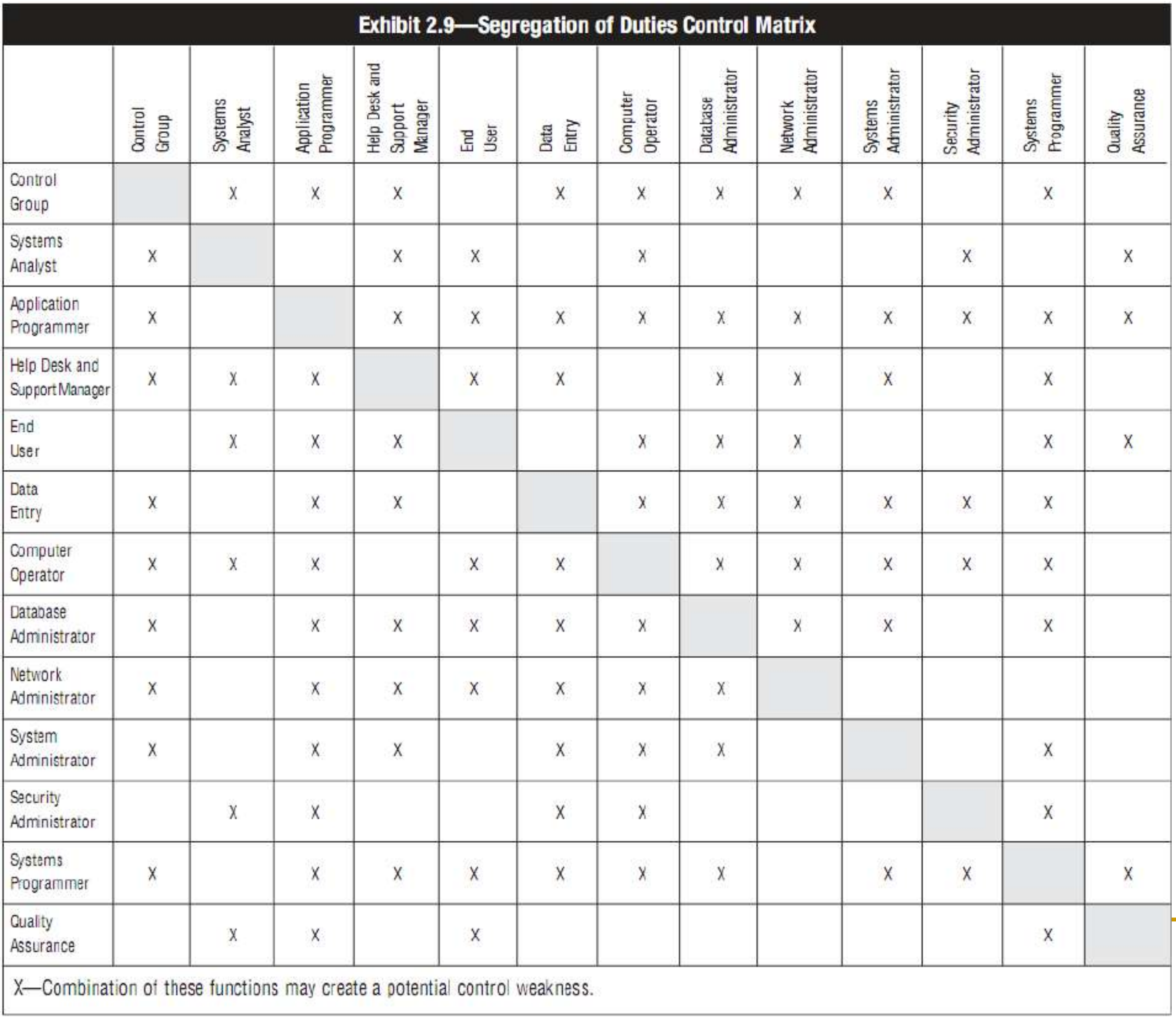


Exhibit 2.9—Segregation of Duties Control Matrix

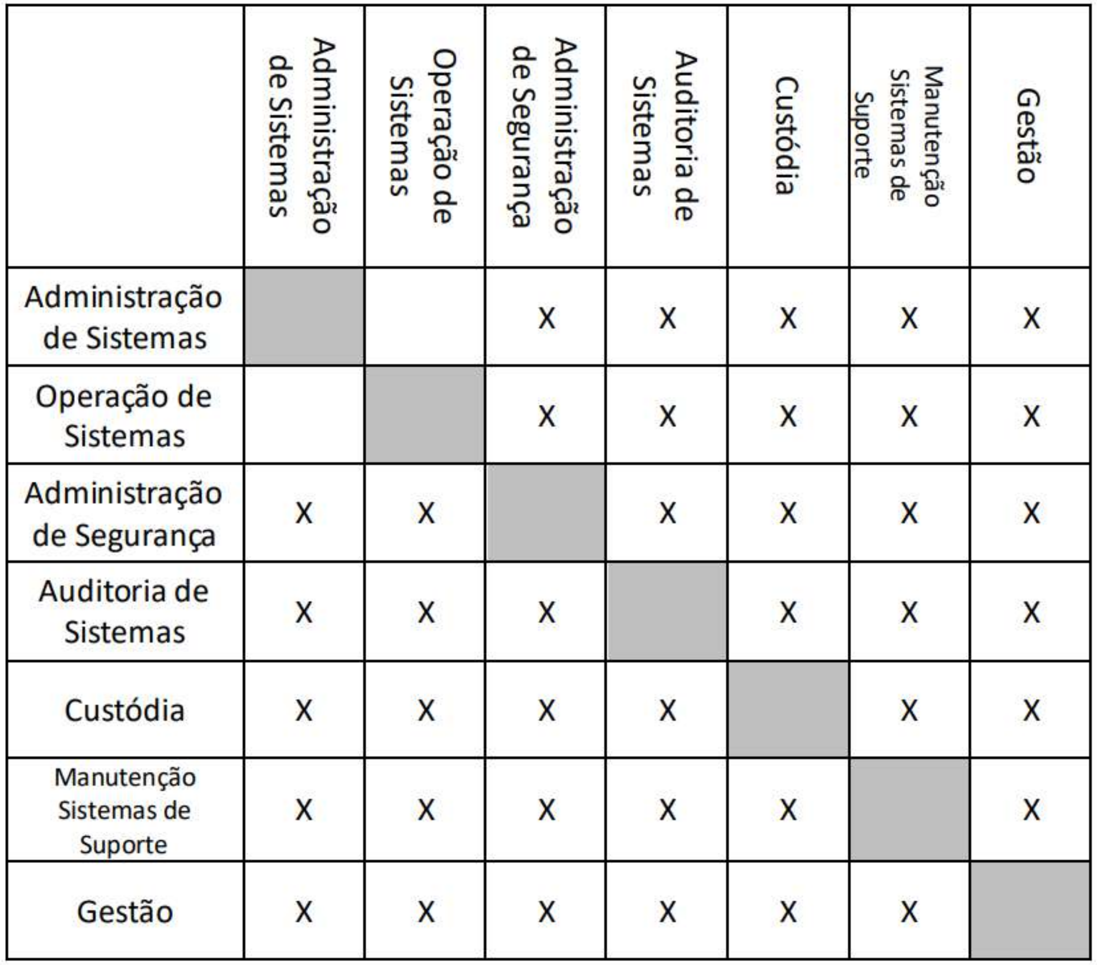
	Control Group	Systems Analyst	Application Programmer	Help Desk and Support Manager	End User	Data Entry	Computer Operator	Database Administrator	Network Administrator	Systems Administrator	Security Administrator	Systems Programmer	Quality Assurance
Control Group		X	X	X		X	X	X	X	X		X	
Systems Analyst	X			X	X		X				X		X
Application Programmer	X			X	X	X	X	X	X	X	X	X	X
Help Desk and Support Manager	X	X	X		X	X		X	X	X		X	
End User		X	X	X			X	X	X			X	X
Data Entry	X		X	X			X	X	X	X	X	X	
Computer Operator	X	X	X		X	X		X	X	X	X	X	
Database Administrator	X		X	X	X	X	X		X	X		X	
Network Administrator	X		X	X	X	X	X	X					
System Administrator	X		X	X		X	X	X				X	
Security Administrator		X	X			X	X					X	
Systems Programmer	X		X	X	X	X	X	X		X	X		X
Quality Assurance		X	X		X							X	

X—Combination of these functions may create a potential control weakness.

X marks an incompatibility

Segregation of duties:

Declaração de Práticas de Certificação da EC do Cartão de Cidadão



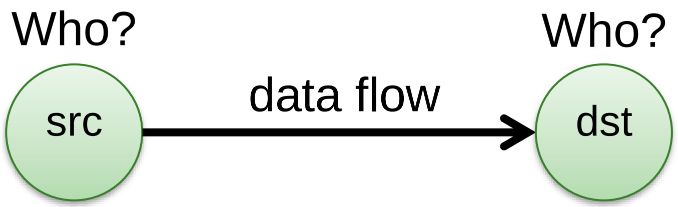
	Administração de Sistemas	Operação de Sistemas	Administração de Segurança	Auditoria de Sistemas	Custódia	Manutenção Sistemas de Suporte	Gestão
Administração de Sistemas			X	X	X	X	X
Operação de Sistemas			X	X	X	X	X
Administração de Segurança	X	X		X	X	X	X
Auditoria de Sistemas	X	X	X		X	X	X
Custódia	X	X	X	X		X	X
Manutenção Sistemas de Suporte	X	X	X	X	X		X
Gestão	X	X	X	X	X	X	

X marks an incompatibility

https://pki.cartaodecidadao.pt/publico/politicas/POL27.CPS_CC.pdf

Information flow models

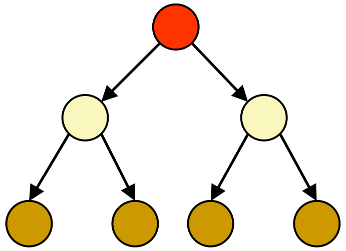
- Authorization is applied to data flows
  - Considering the data flow source and destination
  - Goal: avoid unwanted/dangerous information flows

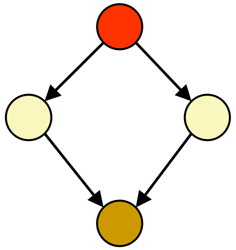


- Src and Dst security-level attributes
  - Information flows should occur only between entities with given security-level (SL) attributes
  - Authorization is given based on the SL attributes

Multilevel security

- Subjects (or roles) act on different security levels
  - Levels do not intersect themselves
  - Levels have some partial order
    - Hierarchy
    - Lattice



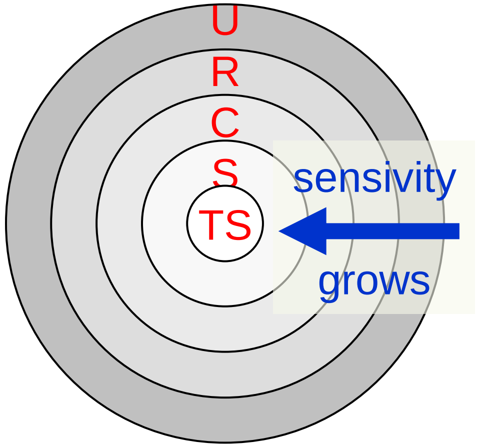


- Levels are used as attributes of subjects and objects
  - Subjects: security level clearance
  - Objects: security classification
- Information flows & security levels
  - Same security level → authorized
  - Different security levels → controlled
    - Authorized or denied on a "need to know" basis

Multilevel security levels: Military / Intelligence organizations

- Typical levels
  - Top secret
  - Secret
  - Confidential
  - Restricted
  - Unclassified

- Portugal (NTE01, NTE04)
  - Muito Secreto
  - Secreto
  - Confidencial
  - Reservado



- EU example
  - EU TOP SECRET
  - EU SECRET
  - EU CONFIDENTIAL
  - EU RESTRICTED
  - EU COUNCIL / COMMISSION

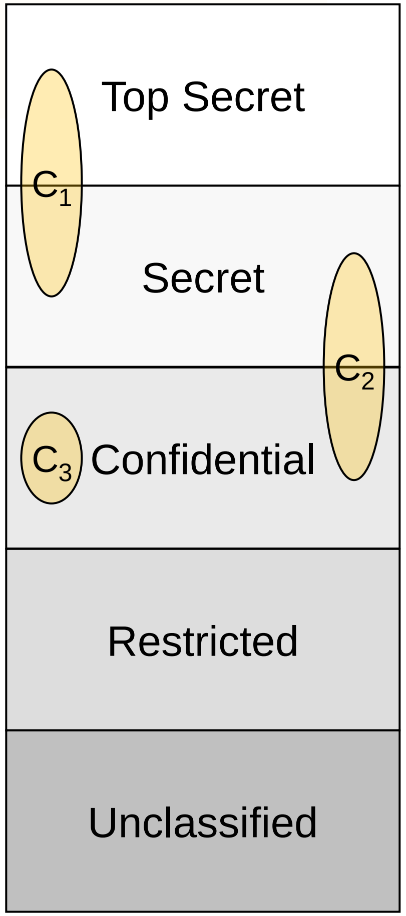
- NATO example:
  - COSMIC TOP SECRET (CTS)
  - NATO SECRET (NS)
  - NATO CONFIDENTIAL (NC)
  - NATO RESTRICTED (NR)

Multilevel security levels: Civil organizations

- Typical levels
  - Restricted
  - Proprietary
  - Sensitive
  - Public

Security categories (or compartments)

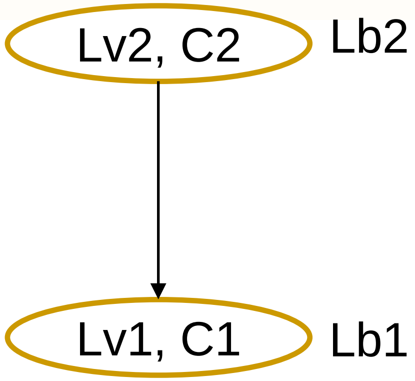
- Self-contained information environments
  - May span several security levels
- Military environments
  - Military branches, military units
- Civil environments
  - Departments, organizational units
- An object can belong to different compartments and have a different security classification in each of them
  - (top-secret, crypto), (secret, weapon)



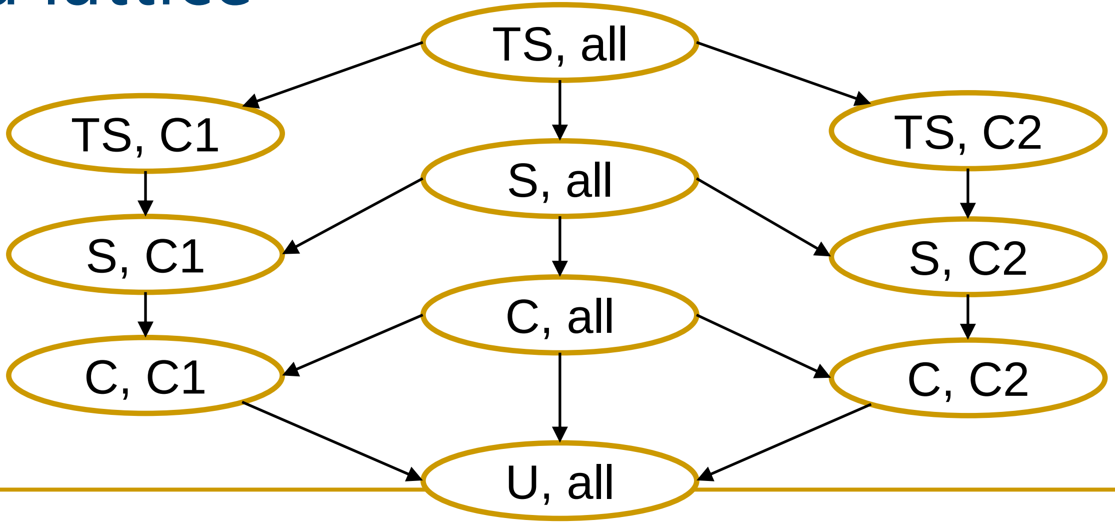
Security labels

- ▷ Label = Category + Level
- ▷ Relative order between labels

  $Lb1 \leq Lb2 \Rightarrow C1 \subseteq C2 \wedge Lv1 \leq Lv2$



- ▷ Labels form a lattice



Bell-La Padula MLS Model

D. Elliott Bell, Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations", MITRE Technical Report 2547, Volume I, 1973

- Access control policy for controlling information flows
  - Addresses data confidentiality and access to classified information
  - Addresses disclosure of classified information
    - Object access control is not enough
    - One needs to restrict the flow of information from a source to authorized destinations

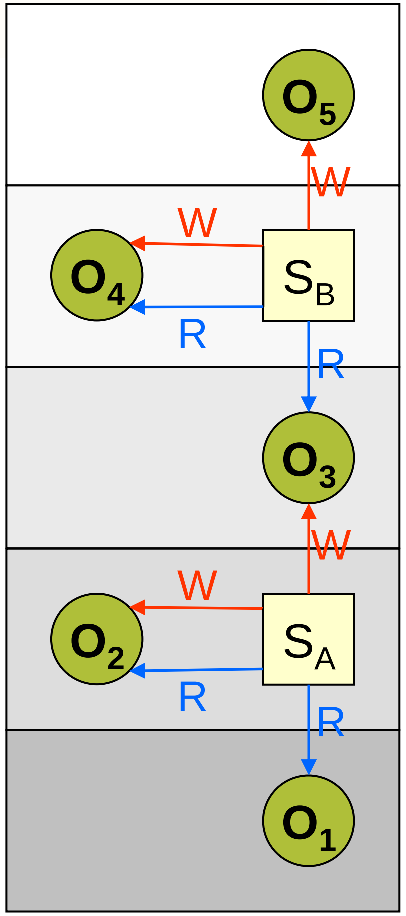
- Uses a state-transition model
  - In each state there are subjects, objects, an access matrix and the current access information
  - State transition rules
  - Security levels and clearances
    - Objects have a security labels
    - Subjects have security clearances
    - Both refer to security levels (e.g. CONFIDENTIAL)

Bell-La Padula MLS Model: Secure state-transition model

- Simple security condition (no read up)
  - S can read O iff L(S) ≥ L(O)

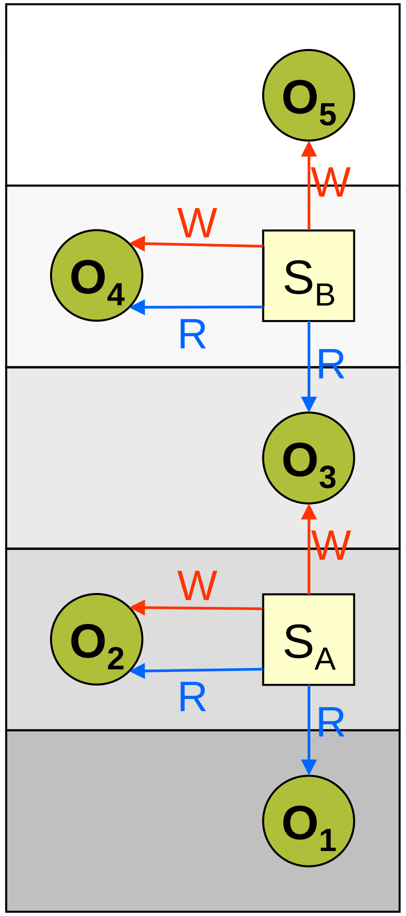
- *-property (no write down)
  - S can write O iff L(S) ≤ L(O)
  - aka confinement property

- Discretionary Security Property
  - DAC-based access control



Bell-La Padula MLS Model:
Secure state-transition model

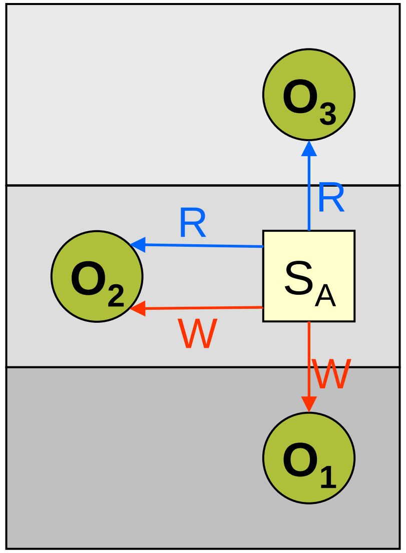
- Strong Star Property
  - S can read O iff L(S) = L(O)
- Tranquility Principle
  - Strong tranquility: S/O levels are static for the entire S/O lifetime
  - Weak tranquility: S/O levels may change if the security *spirit* of the system is not compromised
- Trusted Subjects
  - S can write to lower levels



Biba Integrity Model

K. J. Biba, "Integrity Considerations for Secure Computer Systems", MITRE Technical Report 3153, The Mitre Corporation, April 1977

- Access control policy for controlling information flows
  - For enforcing data integrity control
  - Uses integrity levels, not security levels

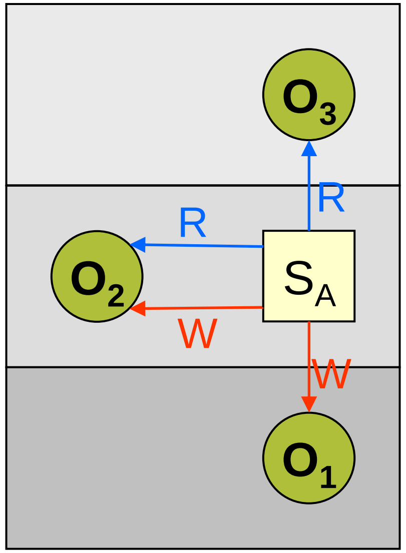


- Similar to Bell-La Padula, with inverse rules
  - Simple Integrity Property (no read down)
    - S can read O iff I(S) ≤ I(O)
  - Integrity *-Property (no write up)
    - S can write O iff I(S) ≥ I(O)

Biba Integrity Model

K. J. Biba, "Integrity Considerations for Secure Computer Systems", MITRE Technical Report 3153, The Mitre Corporation, April 1977

- Access control policy for controlling information flows
  - For enforcing data integrity control
  - Uses integrity levels, not security levels
  - Subjects cannot corrupt objects at higher levels
- Similar to Bell-La Padula, with inverse rules
  - Simple Integrity Property (no read down)
    - S can read O iff I(S) ≤ I(O)
  - Integrity *-Property (no write up)
    - S can write O iff I(S) ≥ I(O)
- Invocation Property
  - S cannot request higher access



Windows mandatory integrity control

- Allows mandatory (priority and critical) access control enforcement prior to evaluate DACLs
  - If access is denied, DACLs are not evaluated
  - If access is allowed, DACLs are evaluated

- Integrity labels
  - Untrusted
  - Low (or AppContainer)
  - Medium (default)
  - Medium Plus
  - High
  - System
  - Protected Process

DACL: discretionary access control list

https://learn.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control

Windows mandatory integrity control

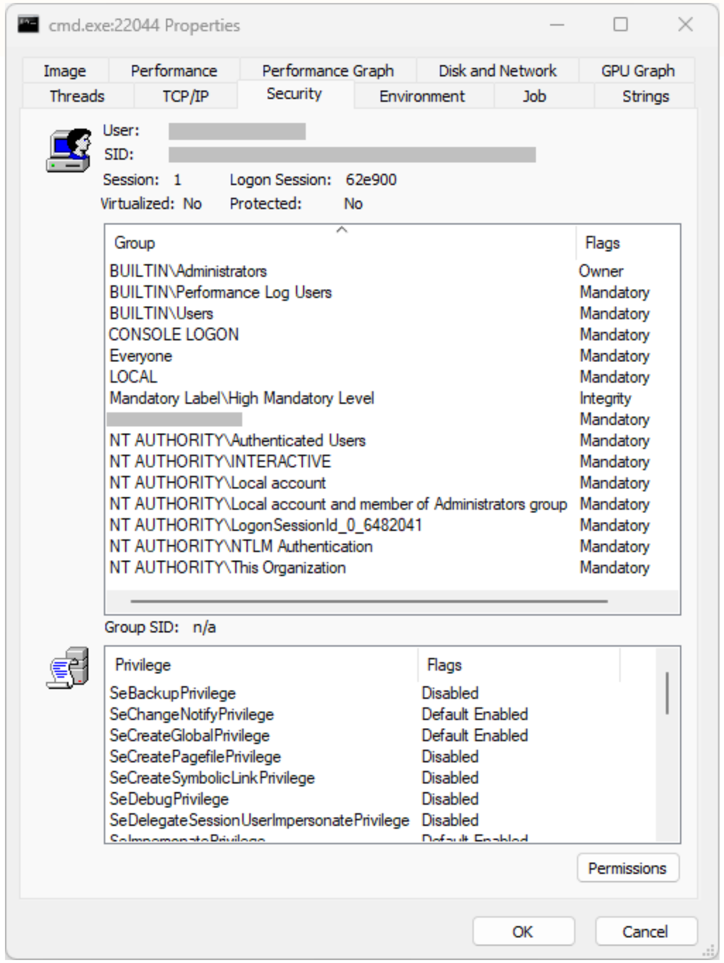
- Users
  - Medium: standard users
  - High: elevated users

- Process integrity level
  - The minimum associated to the owner and the executable file
  - User processes usually are Medium or High
    - Except if executing Low-labeled executables
  - Service processes: High

Windows mandatory integrity control

- Securable objects mandatory label
  - NO_WRITE_UP (default)
  - NO_READ_UP
  - NO_EXECUTE_UP

Windows mandatory integrity control

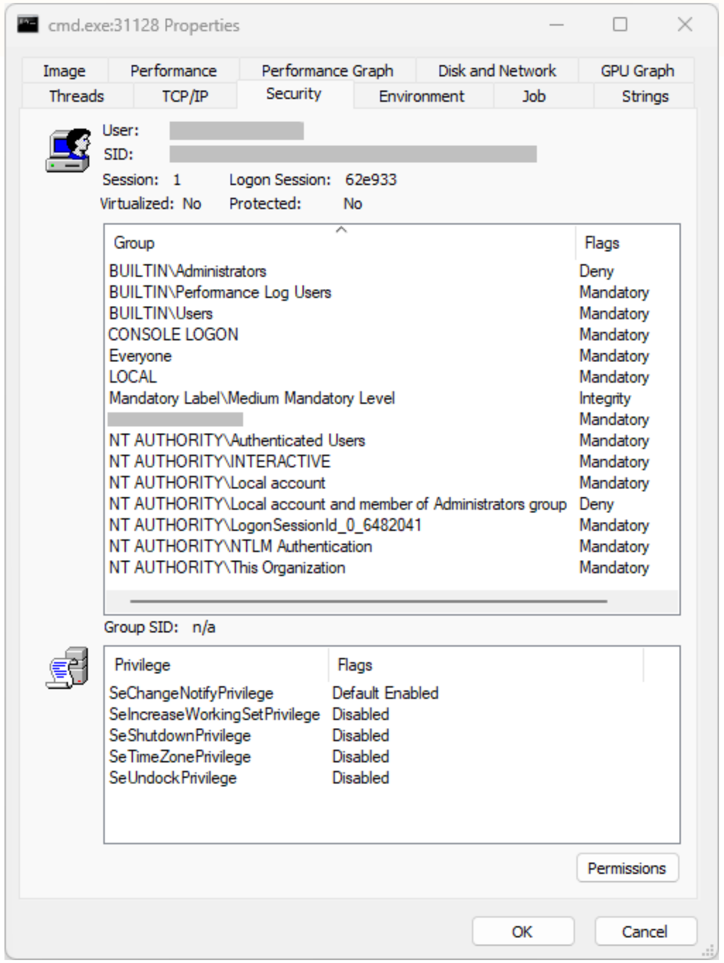


User: [redacted]
SID: [redacted]
Session: 1 Logon Session: 62e900
Virtualized: No Protected: No

Group	Flags
BUILTIN\Administrators	Owner
BUILTIN\Performance Log Users	Mandatory
BUILTIN\Users	Mandatory
CONSOLE LOGON	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Mandatory Label\High Mandatory Level	Integrity
[redacted]	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\Local account	Mandatory
NT AUTHORITY\Local account and member of Administrators group	Mandatory
NT AUTHORITY\LogonSessionId_0_6482041	Mandatory
NT AUTHORITY\NTLM Authentication	Mandatory
NT AUTHORITY\This Organization	Mandatory

Group SID: n/a

Privilege	Flags
SeBackupPrivilege	Disabled
SeChangeNotifyPrivilege	Default Enabled
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeCreateSymbolicLinkPrivilege	Disabled
SeDebugPrivilege	Disabled
SeDelegateSessionUserImpersonatePrivilege	Disabled
SeImpersonatePrivilege	Default Enabled



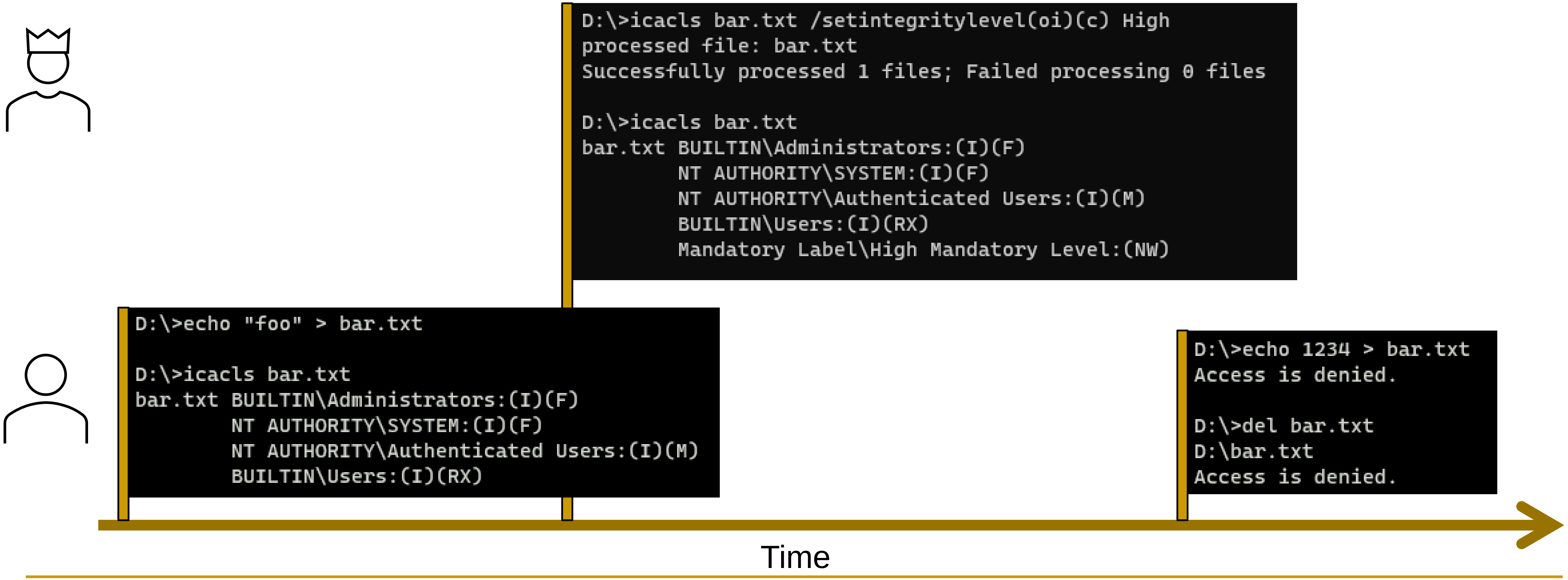
User: [redacted]
SID: [redacted]
Session: 1 Logon Session: 62e933
Virtualized: No Protected: No

Group	Flags
BUILTIN\Administrators	Deny
BUILTIN\Performance Log Users	Mandatory
BUILTIN\Users	Mandatory
CONSOLE LOGON	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Mandatory Label\Medium Mandatory Level	Integrity
[redacted]	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\Local account	Mandatory
NT AUTHORITY\Local account and member of Administrators group	Deny
NT AUTHORITY\LogonSessionId_0_6482041	Mandatory
NT AUTHORITY\NTLM Authentication	Mandatory
NT AUTHORITY\This Organization	Mandatory

Group SID: n/a

Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeIncreaseWorkingSetPrivilege	Disabled
SeShutdownPrivilege	Disabled
SeTimeZonePrivilege	Disabled
SeUndockPrivilege	Disabled

Windows mandatory integrity control



```
D:\>echo "foo" > bar.txt

D:\>icacls bar.txt
bar.txt BUILTIN\Administrators:(I)(F)
        NT AUTHORITY\SYSTEM:(I)(F)
        NT AUTHORITY\Authenticated Users:(I)(M)
        BUILTIN\Users:(I)(RX)
```

```
D:\>icacls bar.txt /setintegritylevel(oi)(c) High
processed file: bar.txt
Successfully processed 1 files; Failed processing 0 files

D:\>icacls bar.txt
bar.txt BUILTIN\Administrators:(I)(F)
        NT AUTHORITY\SYSTEM:(I)(F)
        NT AUTHORITY\Authenticated Users:(I)(M)
        BUILTIN\Users:(I)(RX)
        Mandatory Label\High Mandatory Level:(NW)
```

```
D:\>echo 1234 > bar.txt
Access is denied.

D:\>del bar.txt
D:\bar.txt
Access is denied.
```

Clark-Wilson Integrity Model

D. D. Clark, D. R. Wilson, “A Comparison of Commercial and Military Computer Security Policies”, IEEE Symposium on Security and Privacy, 1987

- Addresses information integrity control
  - Uses the notion of transactional data transformations
  - Separation of duty: transaction certifiers ≠ implementers

- Terminology
  - Data items
    - Constrained Data Item (**CDI**)
      - Can only be manipulated by TPs
    - Unconstrained Data Item (**UDI**)

  - Integrity policy procedures
    - Integrity Verification Procedure (**IVP**)
      - Ensures that all CDIs conform with the integrity specification
    - Transformation Procedure (**TP**)
      - Well-formed transaction
        - Take as input a CDI or a UDI and produce a CDI
      - Must guarantee (via certification) that transforms all possible UDI values to “safe” CDI values

Clark-Wilson Integrity Model: Certification & Enforcement

- Integrity assurance
  - Certification
    - Relatively to the integrity policy
  - Enforcement
- Two sets of rules
  - Certification Rules (C)
  - Enforcement Rules (E)

Clark-Wilson Integrity Model: Certification & Enforcement rules

- Basic rules

  C1: when an IVP is executed, it must ensure that all CDIs are valid

  C2: for some associated set of CDIs, a TP must transform those CDIs from one valid state to another

  E1: the system must maintain a list of certified relations and ensure only TPs certified to run on a CDI change that CDI

- Separation of duty (external consistency)

  E2: the system must associate a user with each TP and set of CDIs. The TP may access CDIs on behalf of the user if authorized

  C3: allowed user-TP-CDI relations must meet “separation of duty” requirements

- Identification gathering

  E3: the system must authenticate every user attempting a TP (on each attempt)

- Audit trail

  C4: all TPs must append to a log enough information to reconstruct operations

- UDI processing

  C5: a TP taking a UDI as input may only perform valid transactions for all possible values of the UDI. The TP will either accept (convert to CDI) or reject the UDI

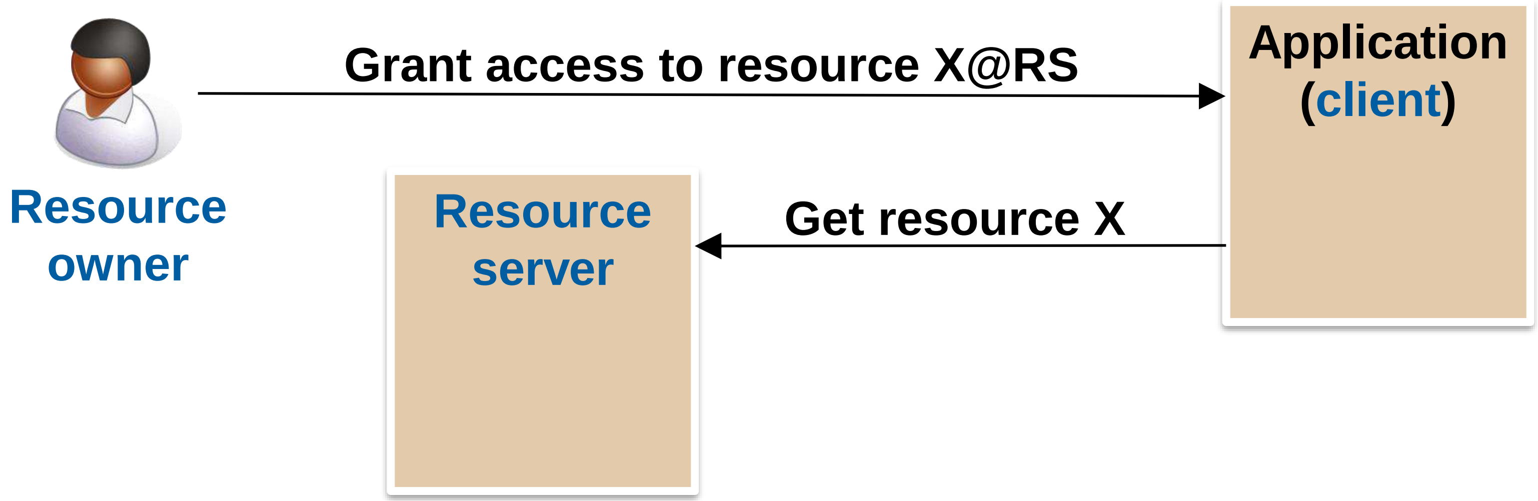
- Certification constraints

  E4: only the certifier of a TP may change the associated list of entities

OAuth 2.0
authorization framework

Goal

- Allow an application to access user resources maintained by a service/server



- Full reference at https://oauth.net/2/

Roles (RFC 6749)

- Resource owner
  - An entity capable of **granting access** to a **protected resource**
  - **End-user**: a resource owner that is a person

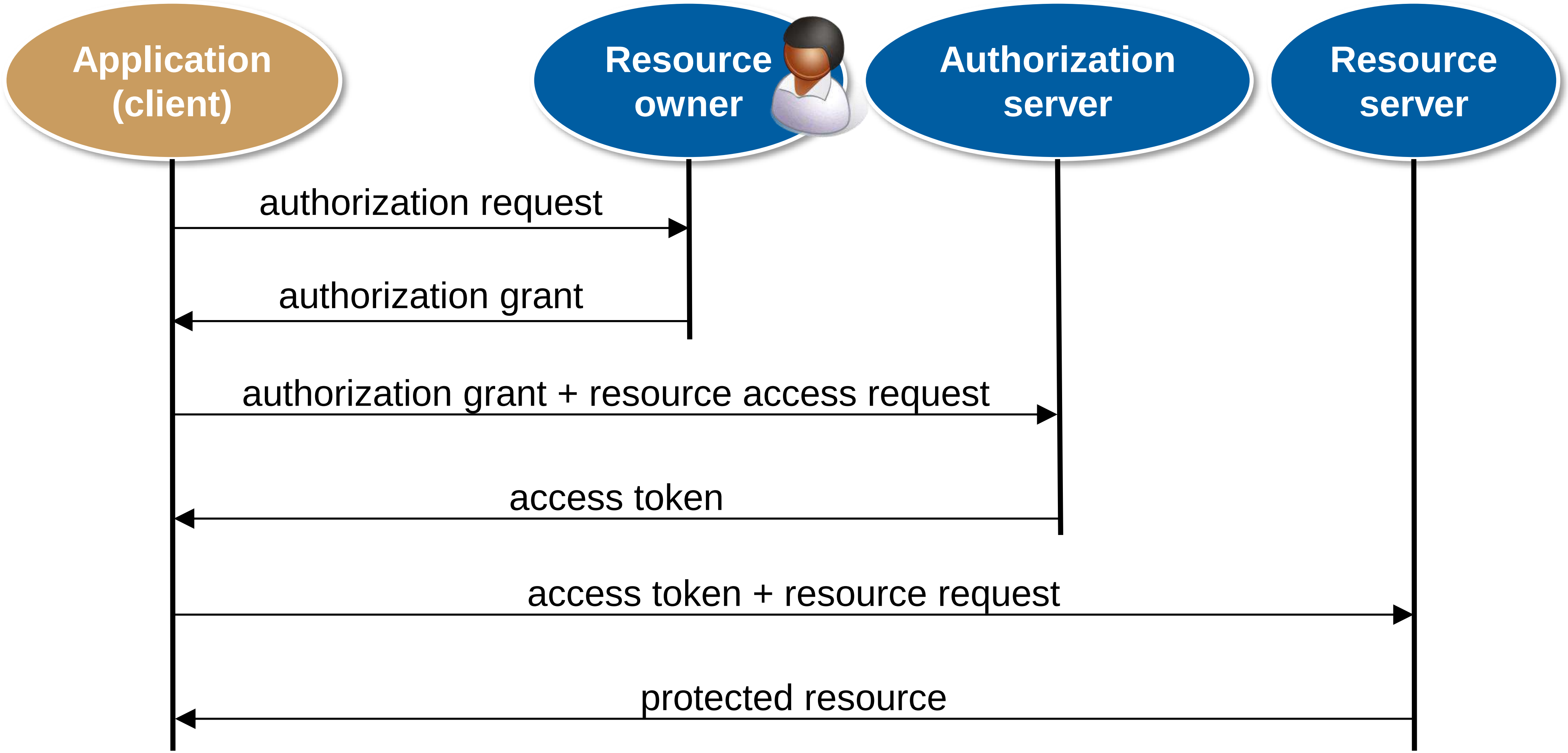
- Resource server
  - The server hosting protected resources
  - Capable of accepting and responding to protected resource requests using **access tokens**

Roles (RFC 6749)

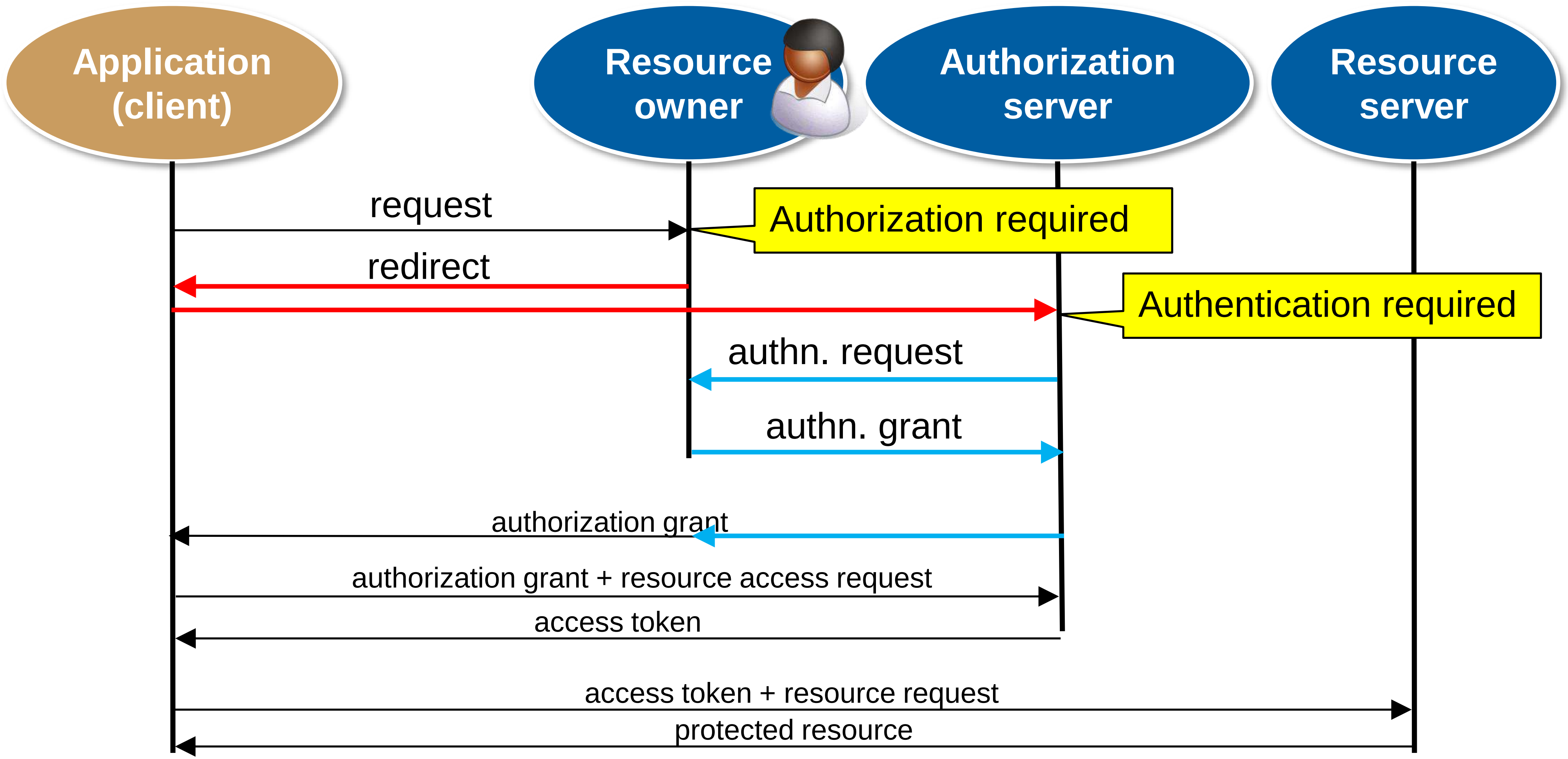
- Client
  - An **application** making requests for protected resources on behalf of the resource owner and with its authorization

- Authorization server
  (aka OAuth server or provider)
  - The server issuing **access tokens** to the client after successfully **authenticating** the resource owner and obtaining its **authorization** for the client to access one of its resources

Abstract protocol flow (RFC 6749)



Common protocol flow



Communication endpoints: Authorization endpoint

- Service provided by the **OAuth server**
  - Authenticates the resource owner (the user)
  - Asks for the delegation of access rights to its protected resources to the client
  - Send an authorization grant to the **redirection endpoint**

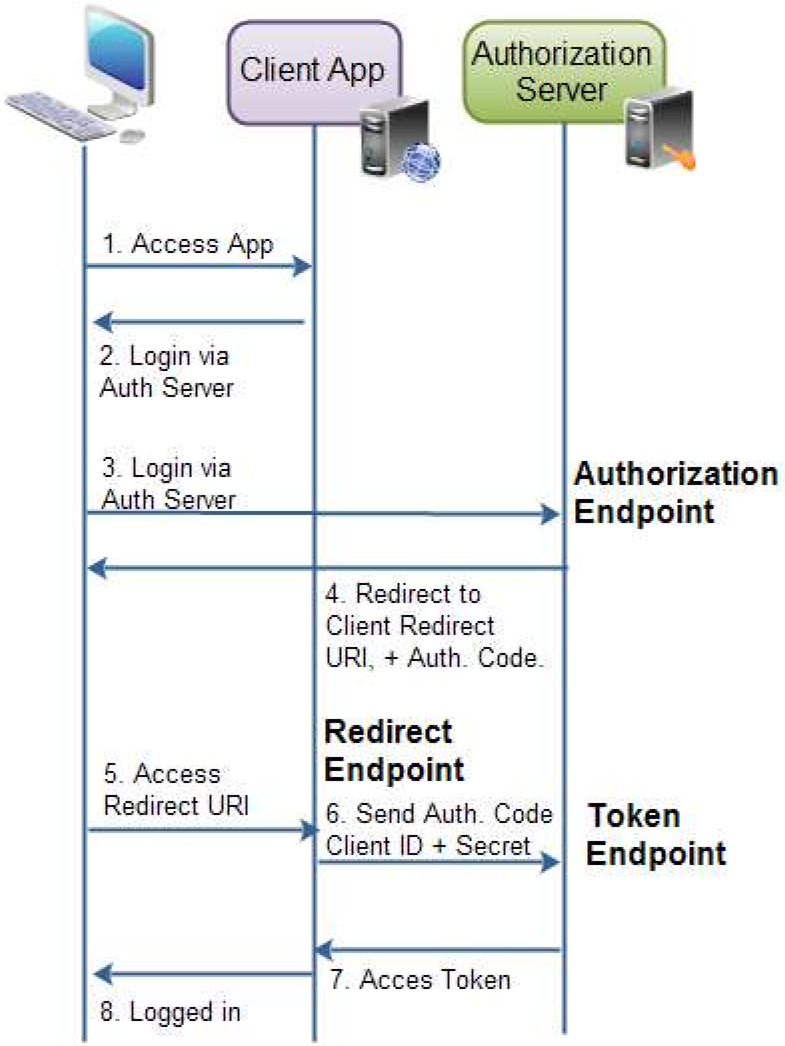
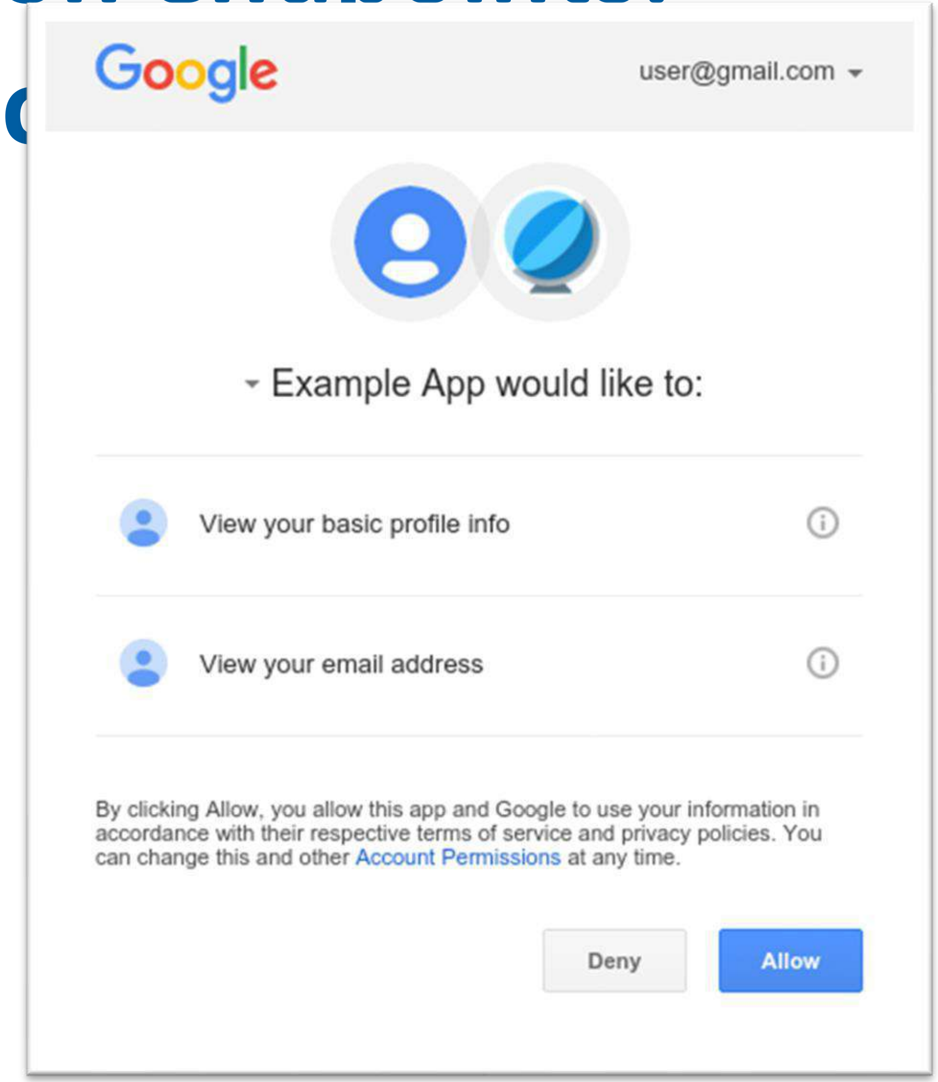


Image: https://jenkov.com/tutorials/oauth2/endpoints.html

Communication endpoints: Authorizati



user@gmail.com

Example App would like to:

View your basic profile info

View your email address

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

Communication endpoints: Token endpoint

- Service provided by the **OAuth server**
  - Produces access tokens given an authorization grant
  - It can also produce refresh tokens
  - Refresh tokens can be used to get new tokens
    - With an authorization grant

- Client authentication
  - ClientID + ClientSecret + HTTP basic authentication

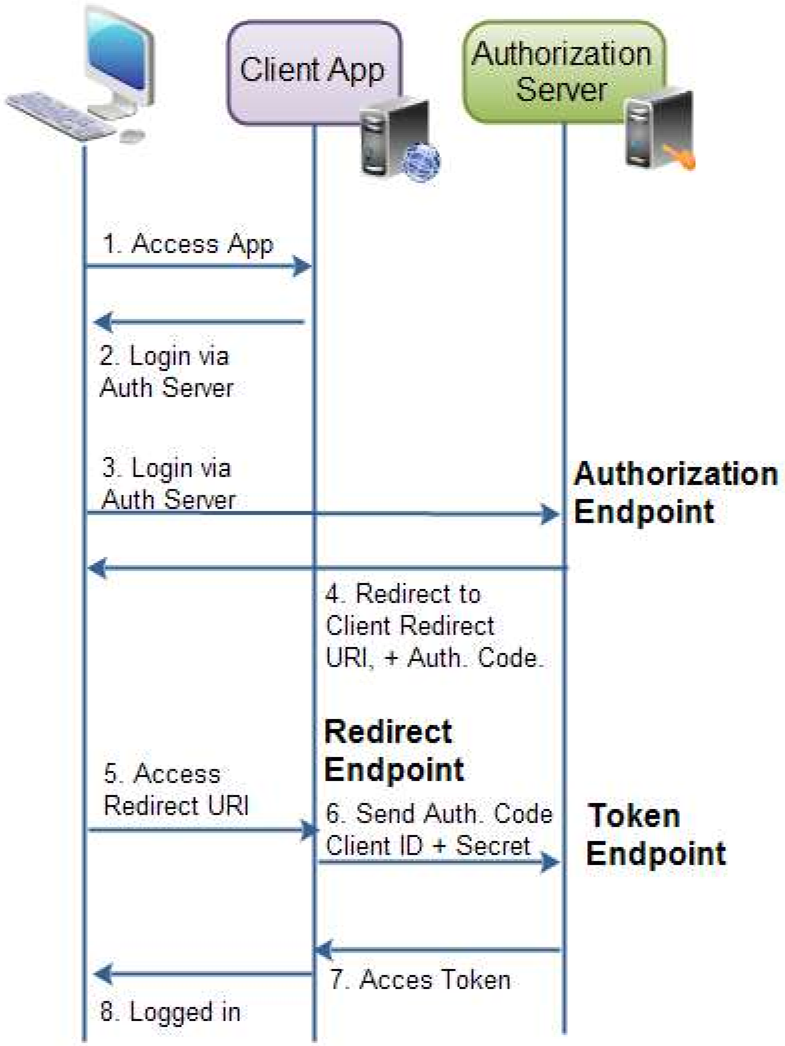


Image: https://jenkov.com/tutorials/oauth2/endpoints.html

Communication endpoints: Redirect endpoint

- Service provided by the client
  - It collects the authorization grant provided by the OAuth server
  - It should be called by the OAuth server using an HTTP redirect

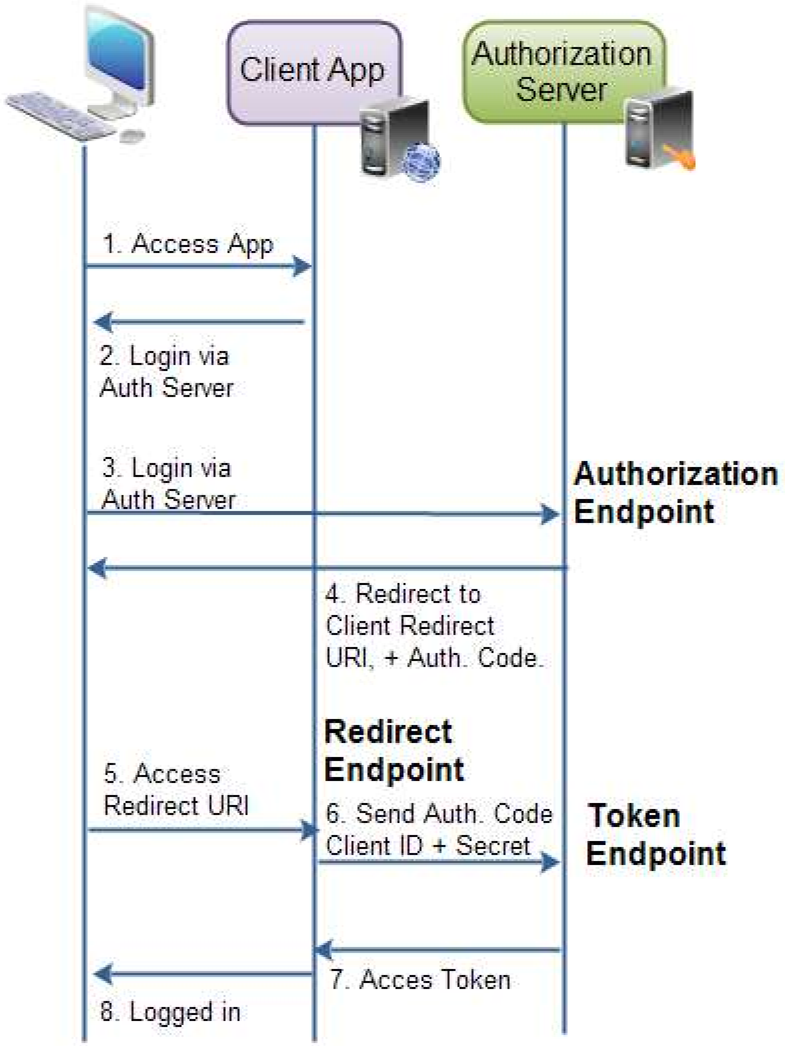


Image: https://jenkov.com/tutorials/oauth2/endpoints.html

Application (client) types

- Type is related with the ability to maintain the confidentiality of client credentials
  - Even from the resource owner

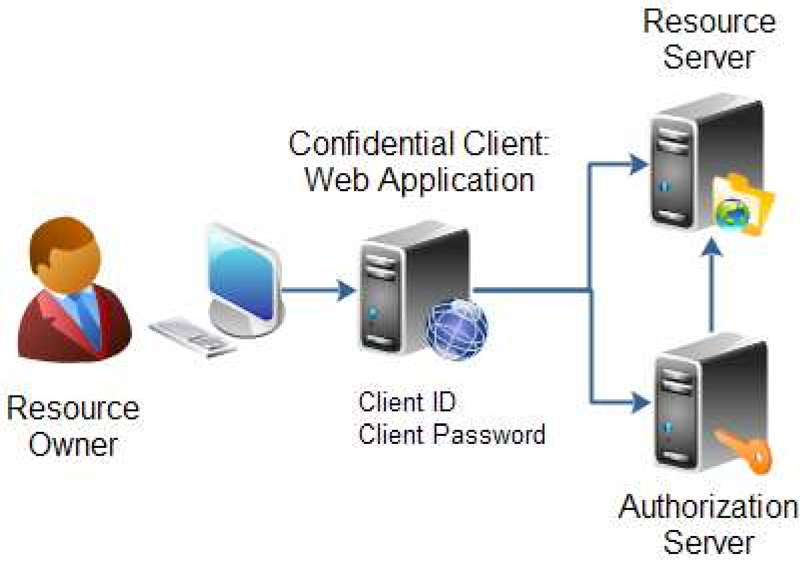
- Confidential
  - Capable
  - e.g. a secure server

- Public
  - Incapable
  - e.g. a web browser-based application, a mobile App

- Different application types will be allowed to execute different flows

Application (client) profiles

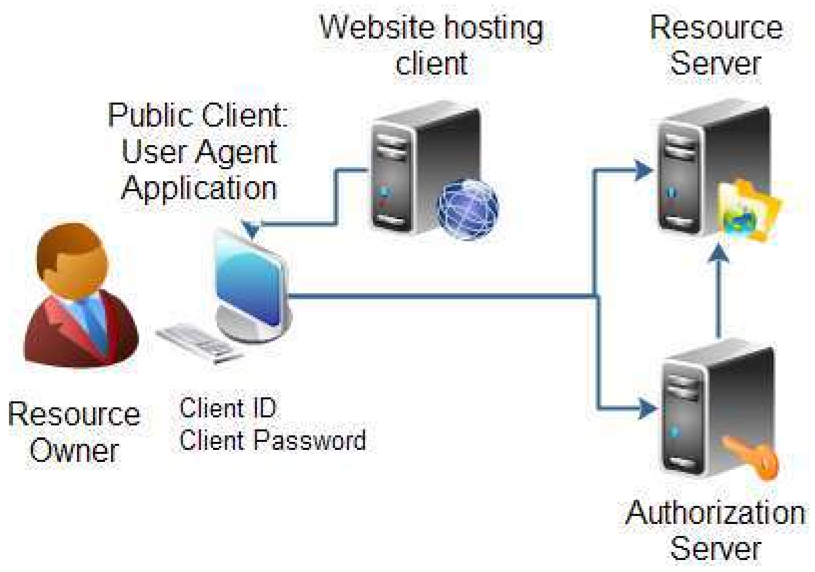
- ▷ Web application
  - Confidential client running on a web server



https://jenkov.com/tutorials/oauth2/client-types.html

Application (client) profiles

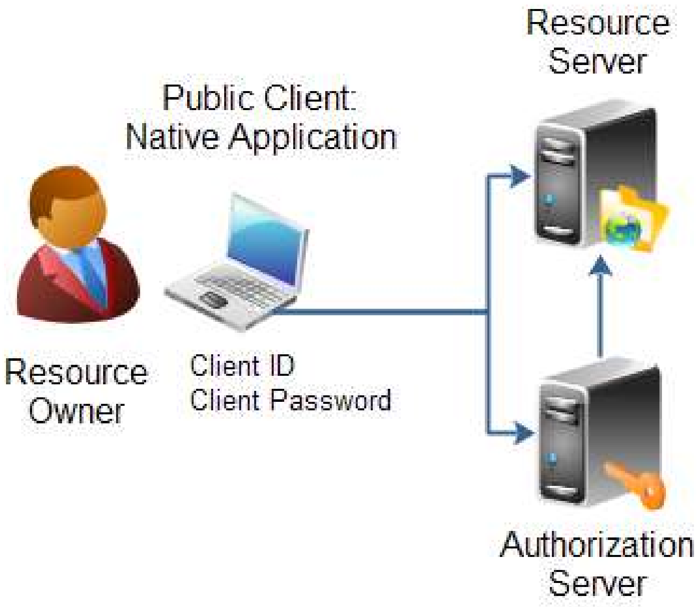
- User-agent based application
  - Public client where the client code runs on a user-agent application
    - e.g. a browser



https://jenkov.com/tutorials/oauth2/client-types.html

Application (client) profiles

- Native application
  - Public client installed and executed on the device used by the resource owner



https://jenkov.com/tutorials/oauth2/client-types.html

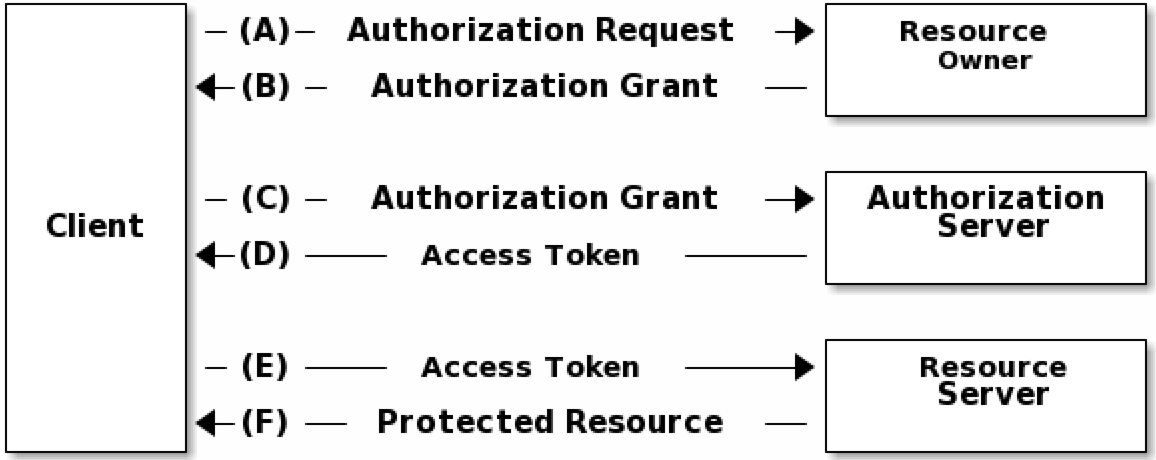
Application (client) registration (in an OAuth server)

- Clients accessing OAuth servers must be previously registered
  - Nevertheless, the standard does not exclude unregistered clients
  - A registered client is given a unique identifier
    - ClientID

- Registration includes both informational, legal and operational information
  - Redirection URLs
  - Acceptance of legal terms
  - Application (client) name, logo, web site, description
  - Client type
  - Client authentication method (for confidential clients)

OAuth tokens: Authorization grant

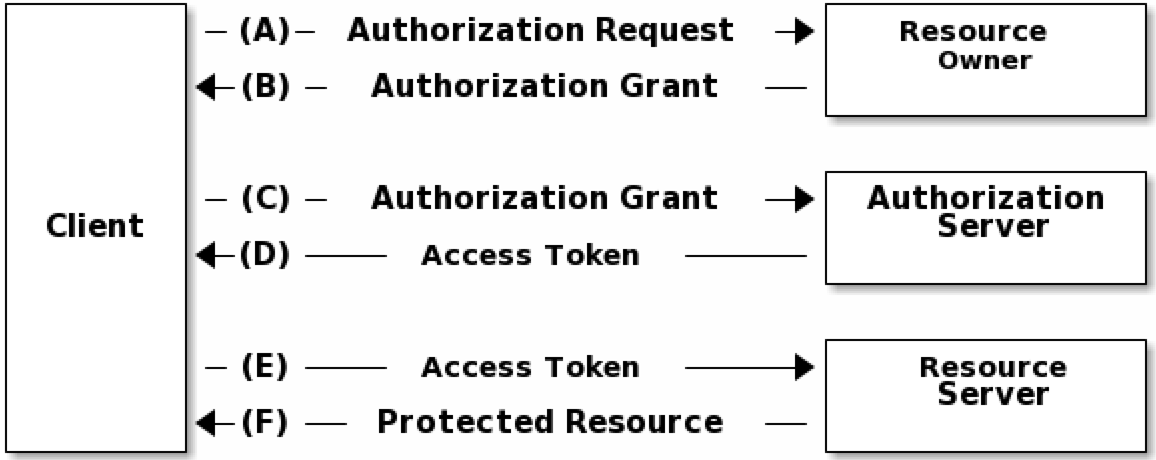
- Created by an OAuth server
  - Upon authenticating a resource owner and getting its consent to grant access to a protected resource
  - An opaque byte blob that makes sense only to its issuer
- Short validity time
  - Just enough to get an access token



OAuth tokens:
Access token

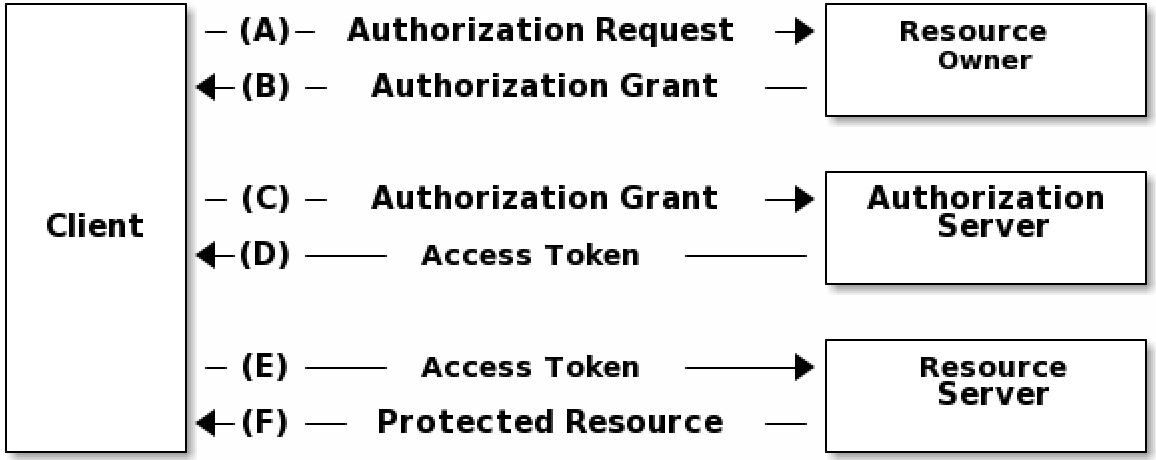
- Created by an OAuth server
  - Upon authenticating a client and receiving an authorization grant
  - An opaque byte blob that makes sense to its issuer and to the resource owner
    - An access capability

- Bearer tokens
  - Clients need to protect their use with HTTPS
  - Clients can handover tokens to others



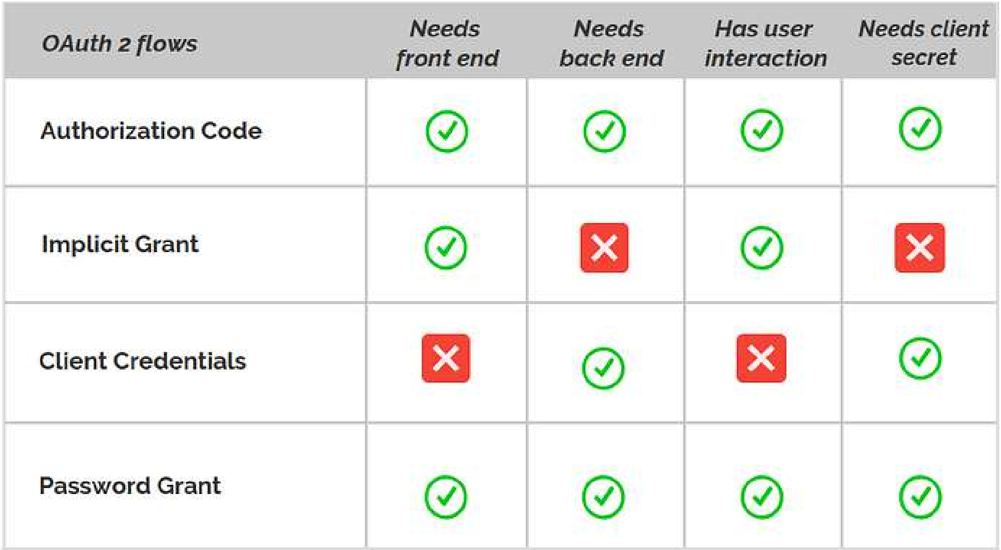
OAuth tokens:
Refresh token

- Created by an OAuth server
  - When creating an access token
  - An opaque byte blob that makes sense only to its issuer
  - It can be used to collect a new access token
    - Still requiring the client authentication
- Bearer tokens
  - Clients need to protect their use with HTTPS
  - Clients can handover tokens to others



OAuth flows

- Authorization code flow
  - 3-legged OAuth
  - Default OAuth flow
  - The most secure
- Implicit flow (grant)
- Resource owner password credentials flow
- Client credentials flow
  - 2-legged flow



OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Authorization Code	✓	✓	✓	✓
Implicit Grant	✓	✗	✓	✗
Client Credentials	✗	✓	✗	✓
Password Grant	✓	✓	✓	✓

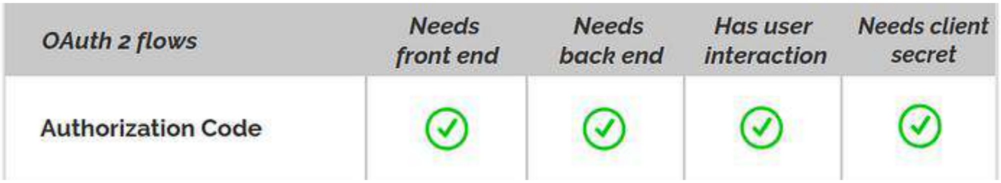
Authorization code flow

- ▷ 3-legged OAuth
  - Enables checking the identity of the 3 involved actors

- ▷ OAuth server authenticates the resource owner
  - Username + password or other means

- ▷ OAuth server authenticates the client
  - ClientID + ClientSecret + HTTP basic authorization

- ▷ Client authenticates the OAuth server
  - Certificate + URL

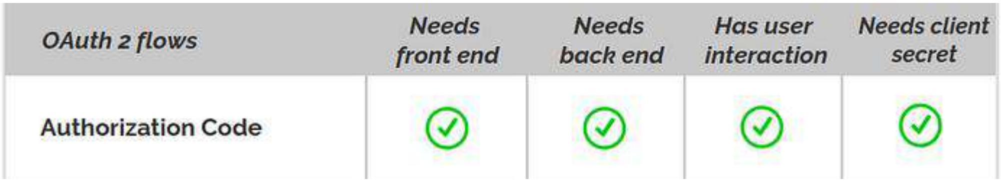


OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Authorization Code	✓	✓	✓	✓

Authorization code flow

- Requirements
  - Confidential application types
  - Secure storage for tokens, ClientID and ClientSecret

- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth

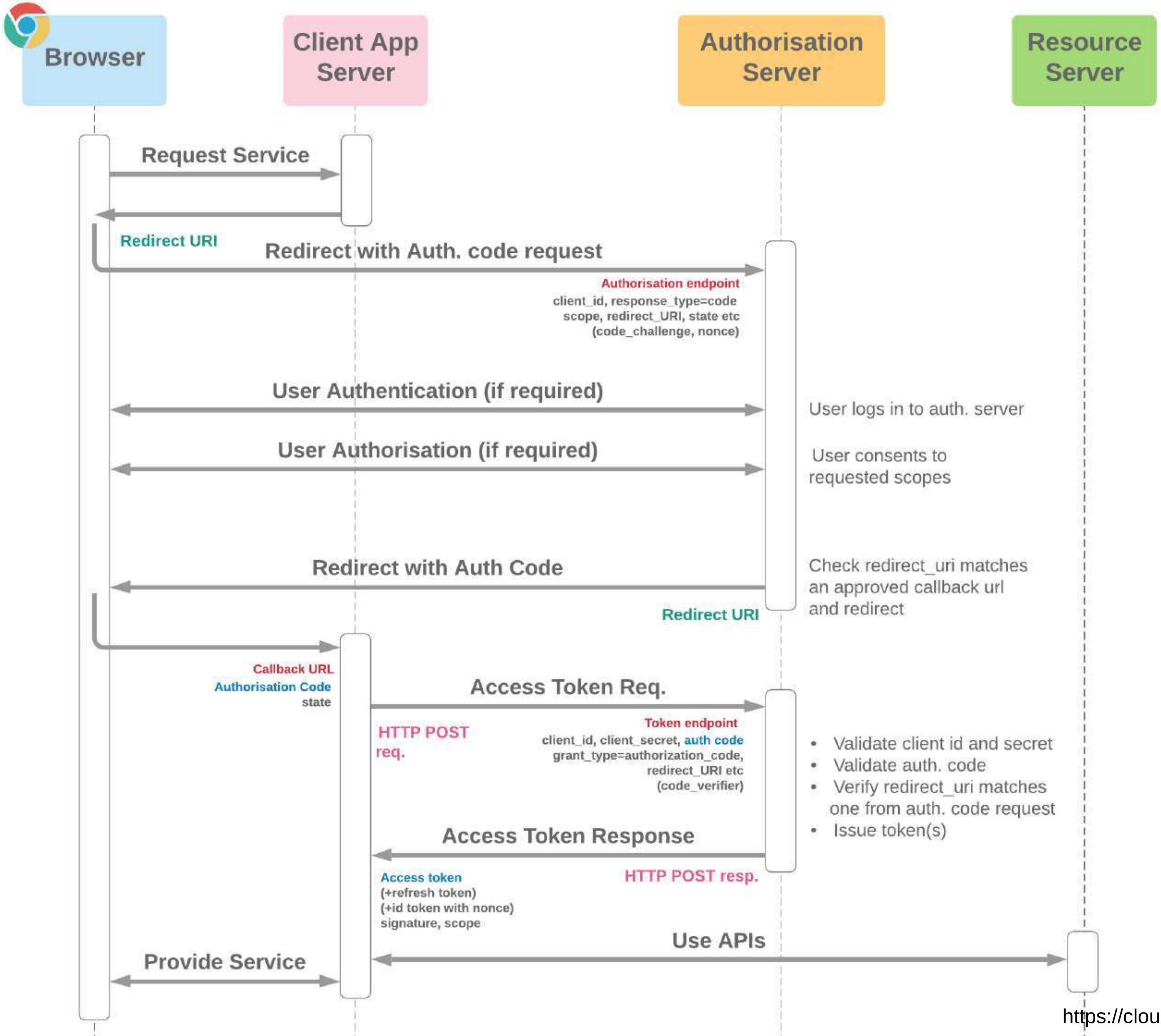


OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Authorization Code	✓	✓	✓	✓

Authorization code flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server redirects the client to the OAuth server
- The OAuth server authenticates the resource owner
  - And sends an authorization grant to the client
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
  - Using its authorization grant
- The client uses again the resource server API to get a resource
  - This time providing an access token

Aut



Browser

Client App Server

Authorisation Server

Resource Server

Request Service

Redirect URI

Redirect with Auth. code request

Authorisation endpoint

client_id, response_type=code
scope, redirect_URI, state etc
(code_challenge, nonce)

User Authentication (if required)

User logs in to auth. server

User Authorisation (if required)

User consents to requested scopes

Redirect with Auth Code

Check redirect_uri matches an approved callback url and redirect

Redirect URI

Callback URL
Authorisation Code
state

Access Token Req.

HTTP POST req.

Token endpoint

client_id, client_secret, auth code
grant_type=authorization_code,
redirect_URI etc
(code_verifier)

- Validate client id and secret
- Validate auth. code
- Verify redirect_uri matches one from auth. code request
- Issue token(s)

Access Token Response

Access token
(+refresh token)
(+id token with nonce)
signature, scope

HTTP POST resp.

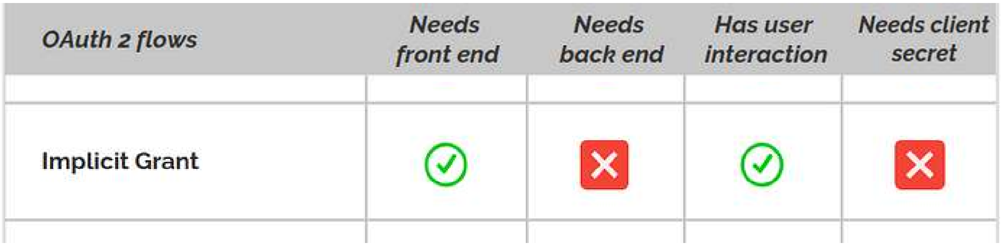
Use APIs

Provide Service

https://cloudsundial.com/salesforce-oauth-flows

Implicit flow

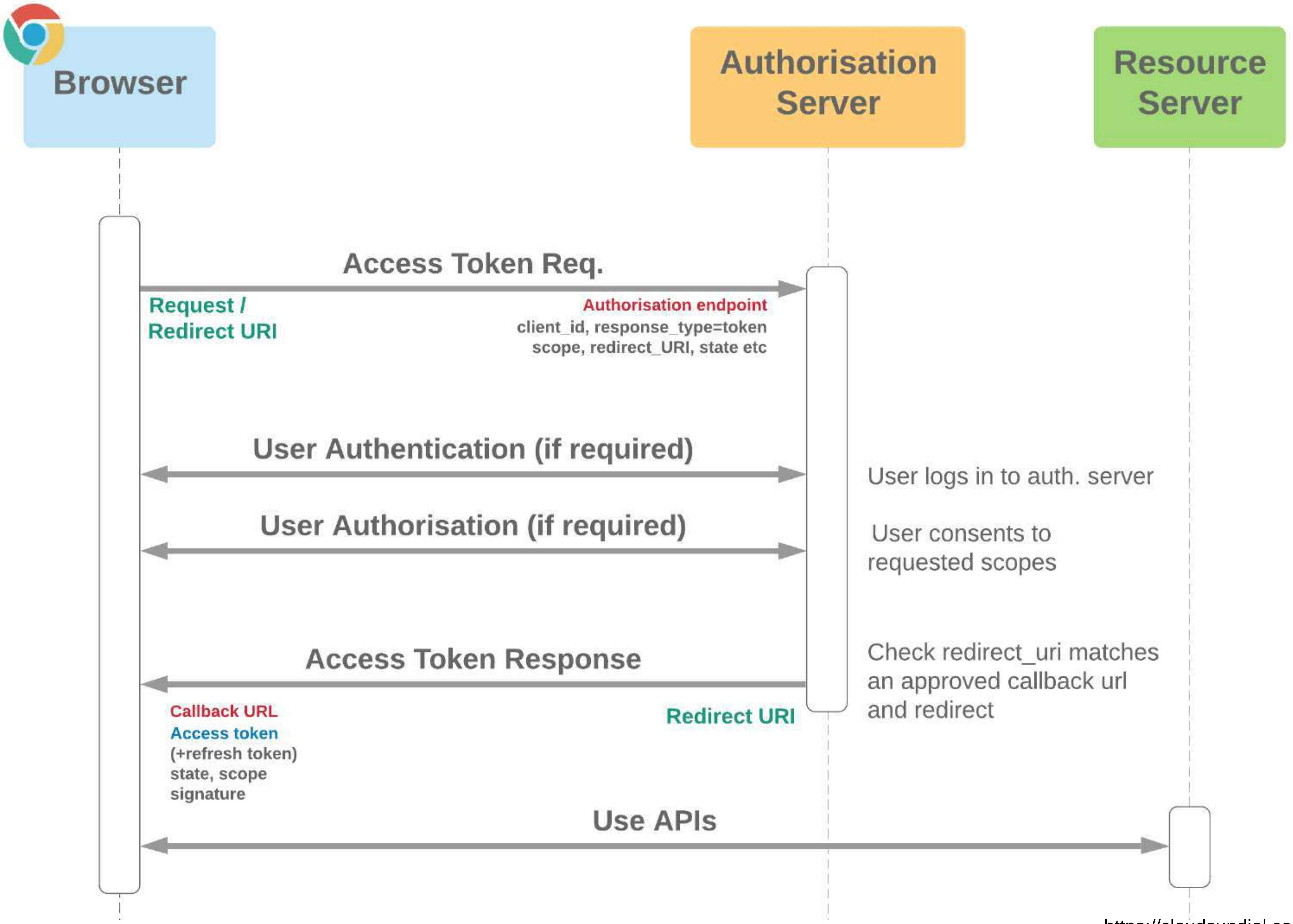
- Requirements
  - Public application types
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID
    - Not regulated by OAuth
- Limitations
  - No client authentication
  - No refresh tokens



OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Implicit Grant	✓	✗	✓	✗

Implicit flow

- Resource owner uses a mobile or client-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server redirects the client to the OAuth server
- The OAuth server authenticates the resource owner
  - And sends an access token to the client
- The client uses again the resource server API to get a resource
  - This time providing an access token



Browser

Authorisation Server

Resource Server

Access Token Req.

Request / Redirect URI

Authorisation endpoint
client_id, response_type=token
scope, redirect_URI, state etc

User Authentication (if required)

User logs in to auth. server

User Authorisation (if required)

User consents to requested scopes

Access Token Response

Check redirect_uri matches an approved callback url and redirect

Callback URL
Access token
(+refresh token)
state, scope
signature

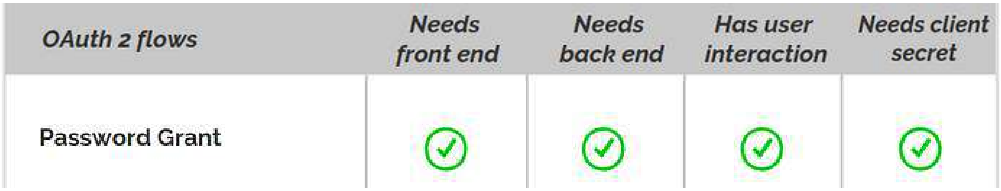
Redirect URI

Use APIs

https://cloudsundial.com/salesforce-oauth-flows

Resource owner password flow

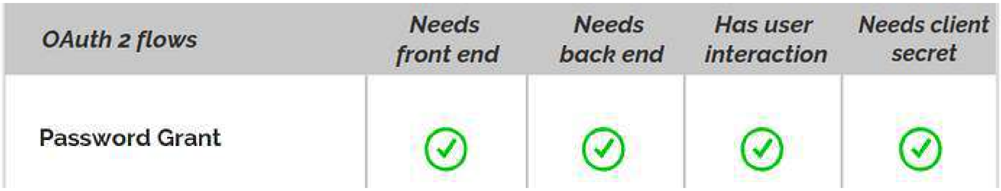
- Requirements
  - Confidential application types
  - Sharing of resource owner credentials with client applications
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth
- Limitations
  - Resource owners need to trust on client applications



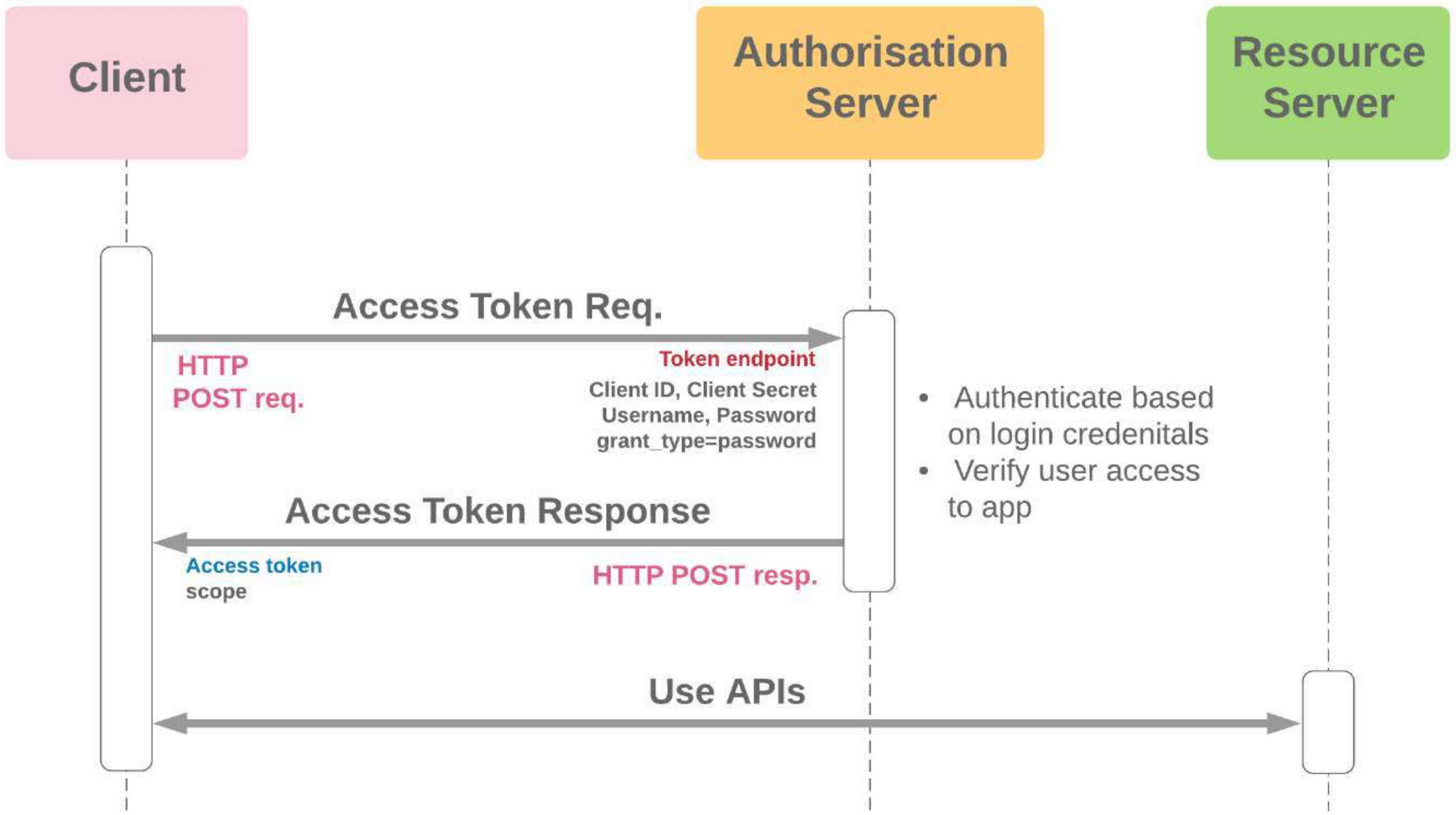
OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Password Grant	✓	✓	✓	✓

Resource owner password flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server requests a token
- The client asks the resource owner for authentication credentials
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
  - Using the resource owner's credentials
  - These should be immediately discarded
- The client uses again the resource server API to get a resource
  - This time providing an access token



OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Password Grant	✓	✓	✓	✓

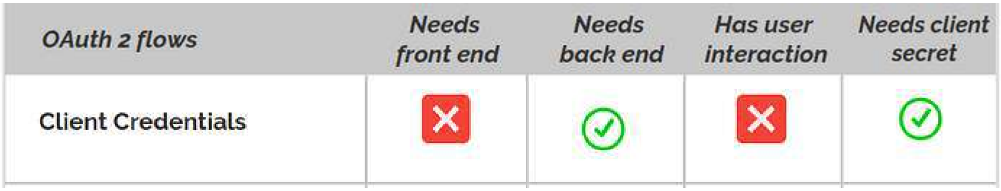

Client
Authorisation Server
Resource Server
Access Token Req.
HTTP POST req.
Token endpoint
Client ID, Client Secret
Username, Password
grant_type=password
• Authenticate based on login credenitals
• Verify user access to app
Access Token Response
Access token
scope
HTTP POST resp.
Use APIs

Client credentials flow

- Requirements
  - Confidential application types
  - Secure storage for tokens, ClientID and ClientSecret

- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth

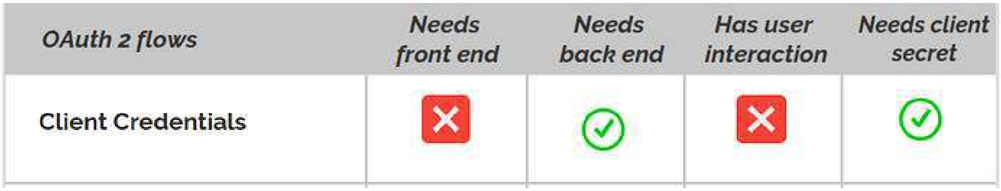
- Limitations
  - No resource owner authentications or authorizations



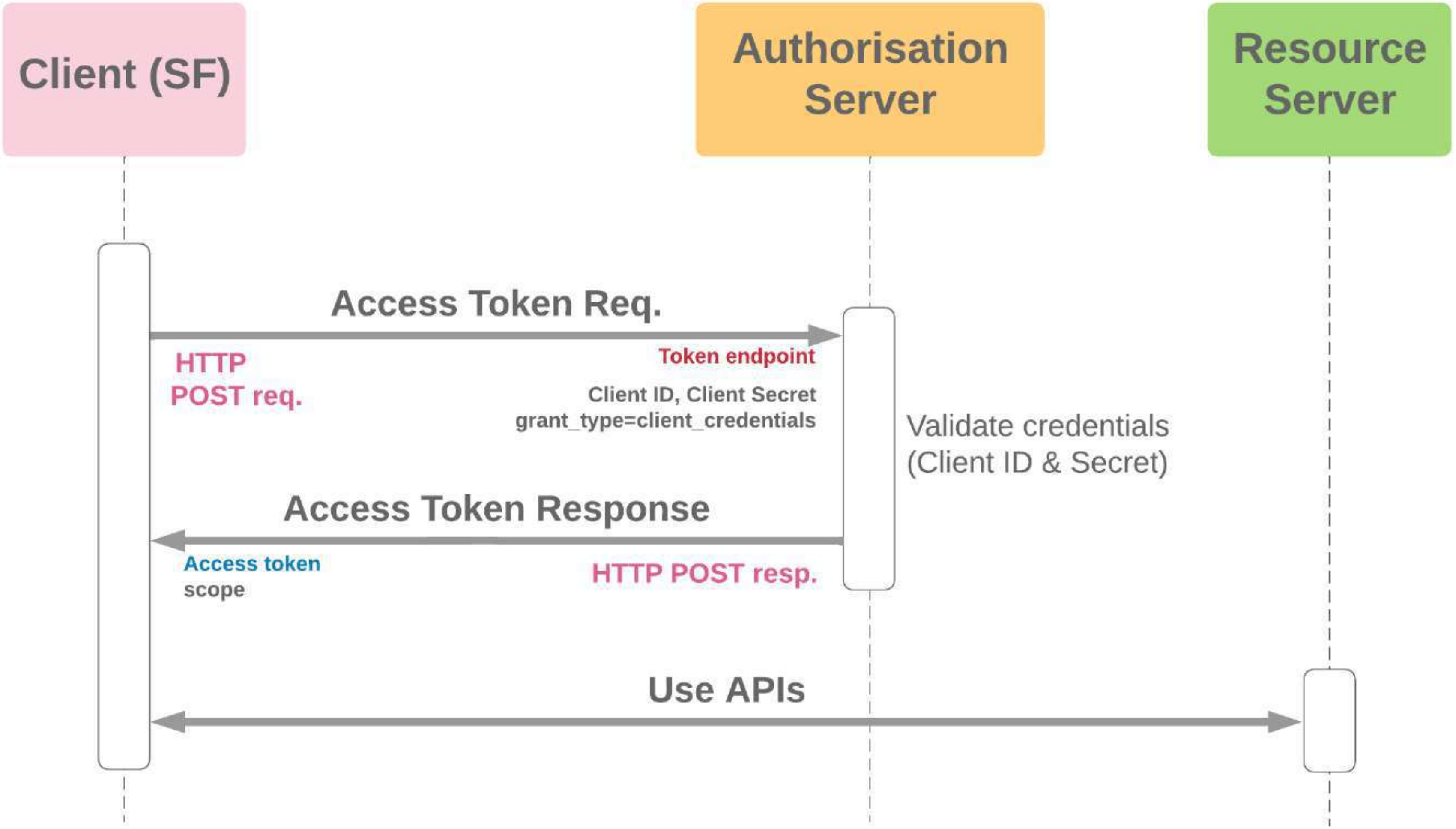
OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Client Credentials	☒	✓	☒	✓

Client credentials flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server requests a token
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
- The client uses again the resource server API to get a resource
  - This time providing an access token



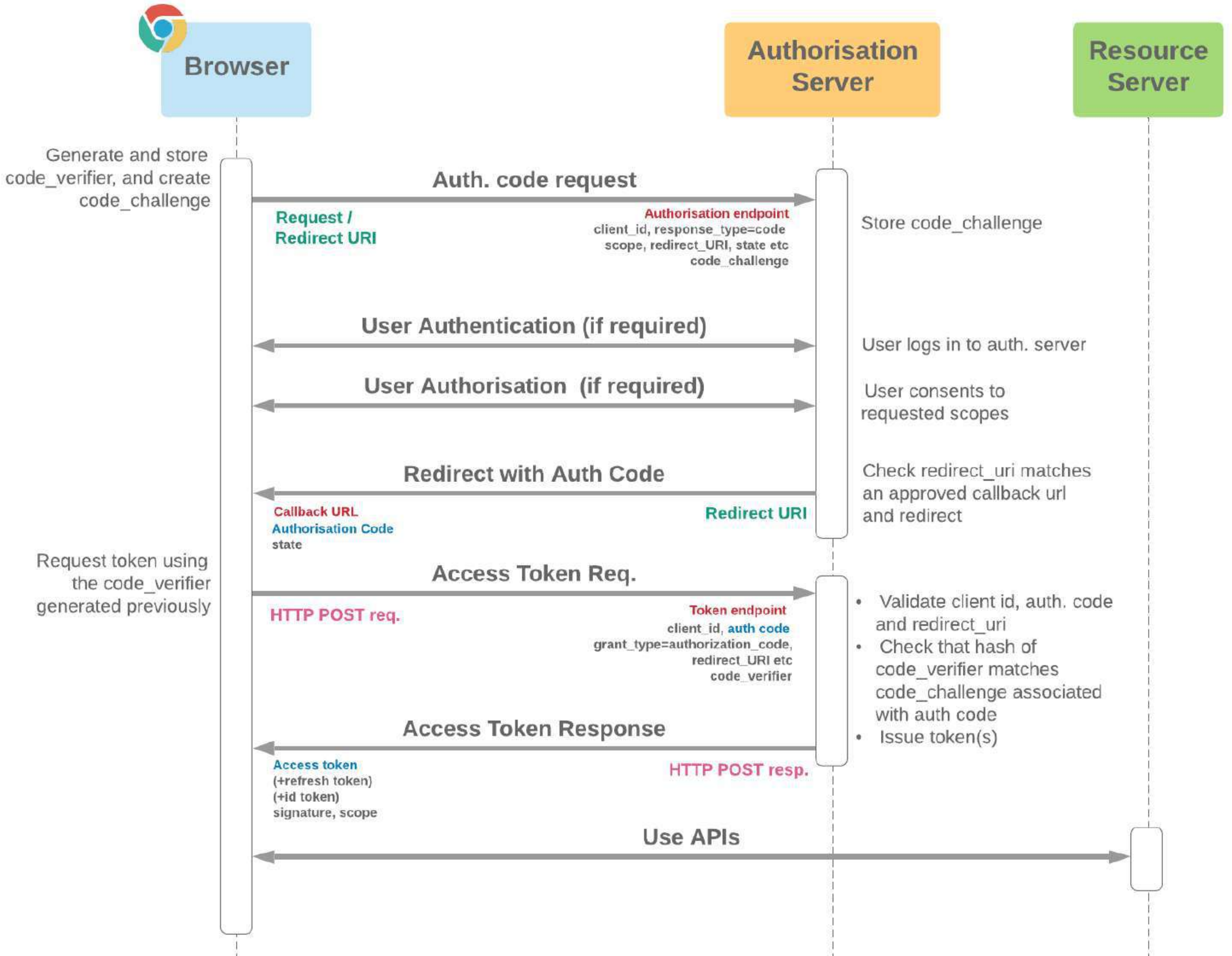
OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Client Credentials	✗	✓	✗	✓


Client (SF)
Authorisation Server
Resource Server
Access Token Req.
HTTP POST req.
Token endpoint
Client ID, Client Secret
grant_type=client_credentials
Validate credentials (Client ID & Secret)
Access Token Response
Access token
scope
HTTP POST resp.
Use APIs

Proof Key for Code Exchange (PKCE, RFC 7636)

- Binds authorization grants to their requesters
  - Using a Code Challenge
    - A digest of a Code Verifier
    - A bit commitment
  - Cannot the used by eavesdroppers

- The requester is required to demonstrate the ownership of the authorization grant when fetching the access token
  - Providing the Code Verifier



Browser

Authorisation Server

Resource Server

Generate and store code_verifier, and create code_challenge

Auth. code request

Request / Redirect URI

Authorisation endpoint
client_id, response_type=code
scope, redirect_URI, state etc
code_challenge

Store code_challenge

User Authentication (if required)

User logs in to auth. server

User Authorisation (if required)

User consents to requested scopes

Redirect with Auth Code

Callback URL
Authorisation Code
state

Redirect URI

Check redirect_uri matches an approved callback url and redirect

Request token using the code_verifier generated previously

Access Token Req.

HTTP POST req.

Token endpoint
client_id, auth code
grant_type=authorization_code,
redirect_URI etc
code_verifier

- Validate client id, auth. code and redirect_uri
- Check that hash of code_verifier matches code_challenge associated with auth code
- Issue token(s)

Access Token Response

Access token
(+refresh token)
(+id token)
signature, scope

HTTP POST resp.

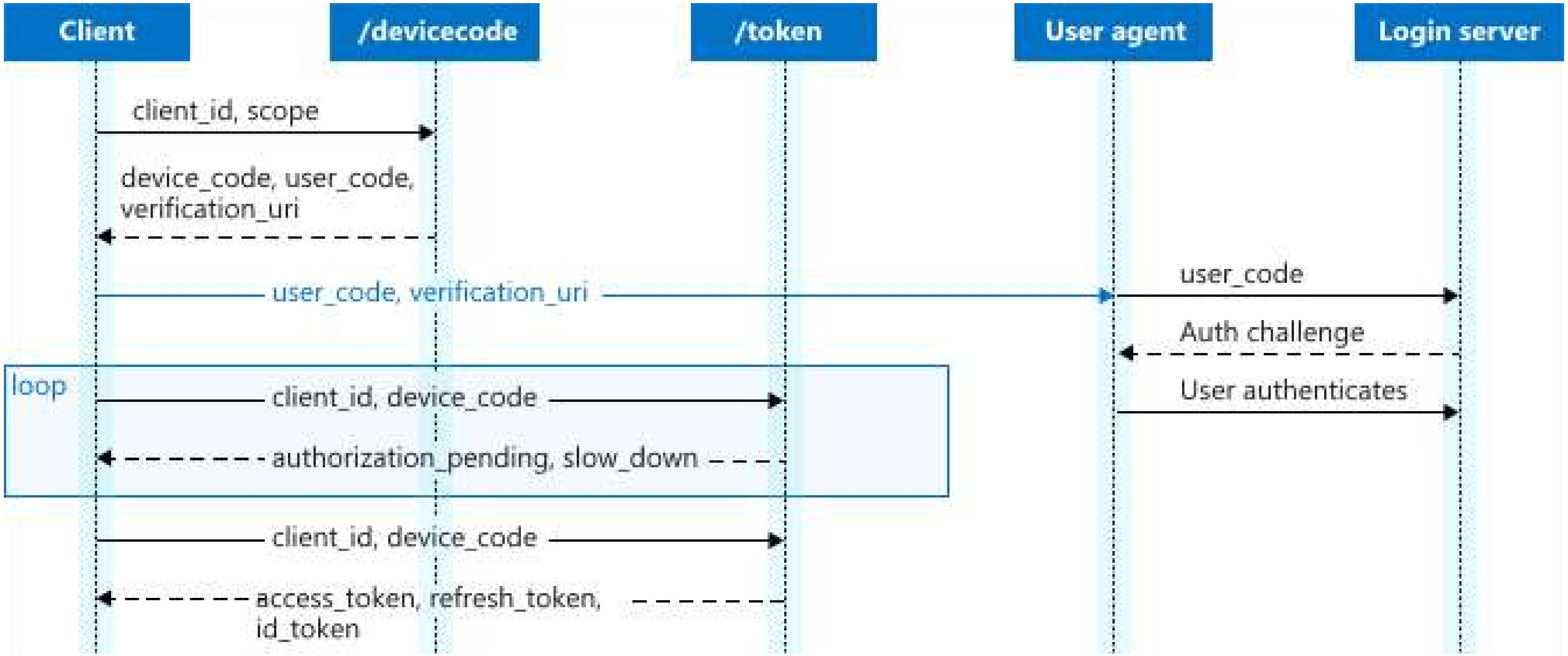
Use APIs

Device authorization grant (RFC 8628)

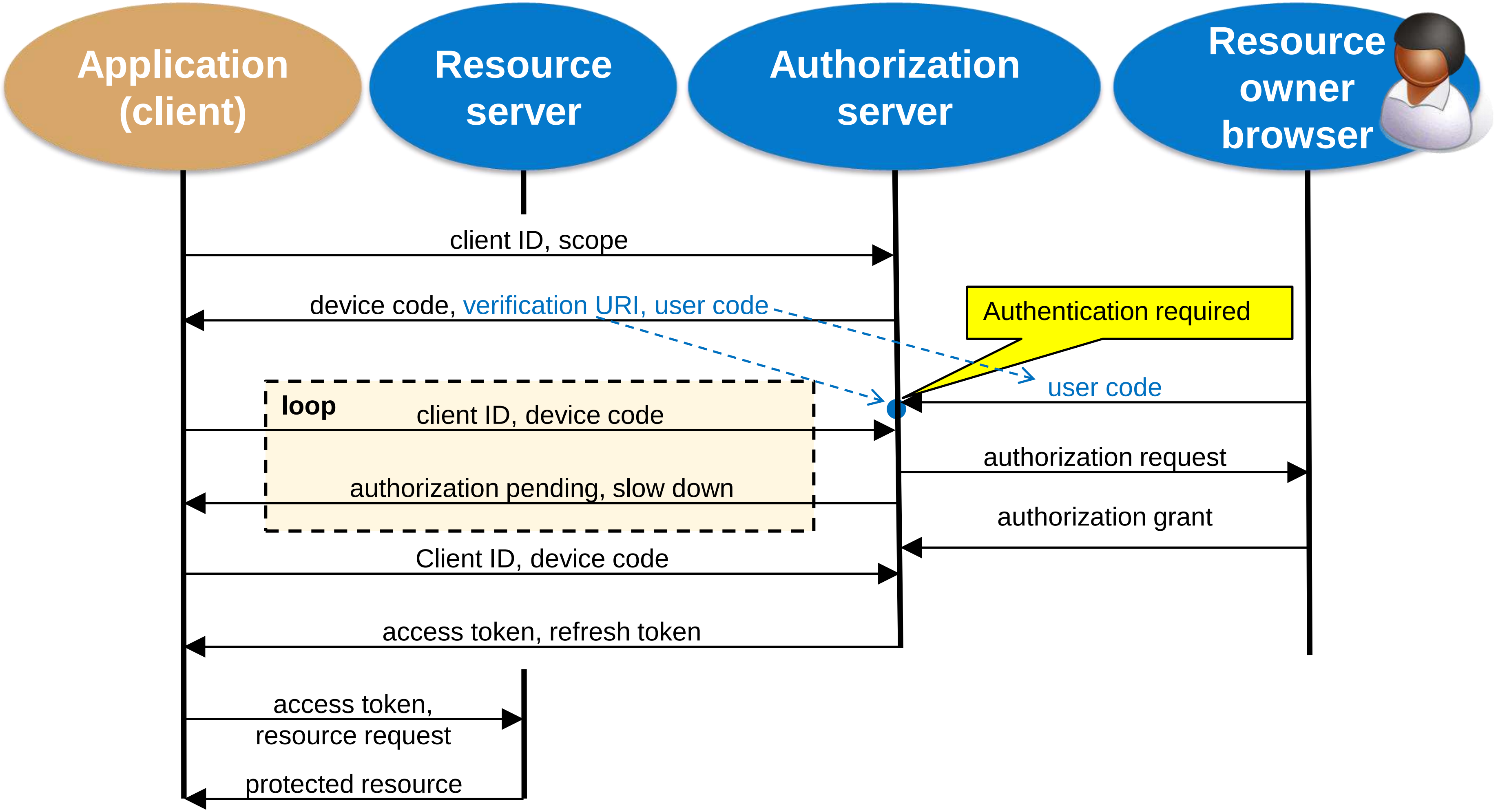
- In some cases the user is using a device with no browser to interact with a OAuth client
  - No HTTP redirections to Authorization server and back to client
  - No user interface
    - To authenticate the user
    - To review and authorize request

- Solution
  - Use a second device to perform the user authentication and to grant the authorization
    - e.g. mobile phone, tablet, etc.
  - Client fetches the access token from the Authorization server
    - Possibly with a refresh token

Device authorization grant (RFC 8628)



Actual protocol flow



Linux security mechanisms

Mechanisms

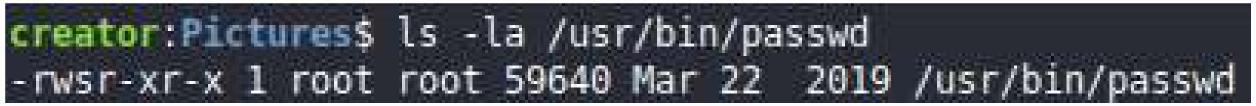
- Capabilities
- cgroups (control groups)
- LSM (Linux Security Modules)

Linux management privileges

- Initial UNIX philosophy
  - Privileged processes (UID = 0)
    - Bypass all kernel permission checks
  - Unprivileged processes (UID ≠ 0)
    - Subject to permission checking based on their credentials
    - Effective UID, effective GID, secondary group list

Unix file protection ACLs: Special protection bits

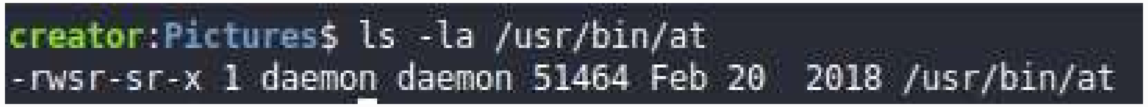
- Set-UID bit



```
creator:Pictures$ ls -la /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Mar 22  2019 /usr/bin/passwd
```

  - Is used to change the UID of processes executing the file

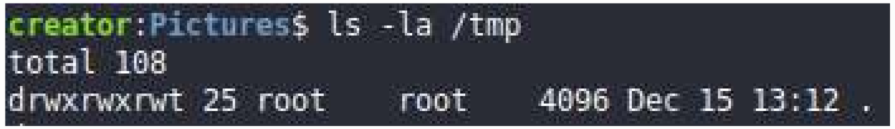
- Set-GID bit



```
creator:Pictures$ ls -la /usr/bin/at
-rwsr-sr-x 1 daemon daemon 51464 Feb 20  2018 /usr/bin/at
```

  - Is used to change the UID of processes executing the file

- Sticky bit



```
creator:Pictures$ ls -la /tmp
total 108
drwxrwxrwt 25 root    root    4096 Dec 15 13:12 .
```

  - Hint to keep the file/directory as much as possible in memory cache

Privilege elevation: Set-UID mechanism

- Change the effective UID of a process running a program stored on a Set-UID file
  - If a program file is owned by UID X and the set-UID bit of its ACL is set, then it will be executed in a process with UID X
    - Independently of the UID of the subject that executed the program

- Allows normal users to execute privileged tasks encapsulated in administration programs
  - Change the user's password (passwd)
  - Change to super-user mode (su, sudo)
  - Mount devices (mount)

Privilege elevation:
Set-UID mechanism (cont.)

- Effective UID / Real UID
  - Real UID (rUID) is the UID of the process creator
    - App launcher
  - Effective UID (eUID) is the UID of the process
    - The one that really matters for defining the rights of the process
    - eUID may differ from rUID

- UID change
  - Ordinary application
    - eUID = rUID = UID of process that executed **exec**
    - eUID cannot be changed (unless = 0)
  - Set-UID application
    - eUID = UID of **exec**’d application file, rUID = initial process UID
    - eUID can revert to rUID
  - rUID cannot change

Privilege elevation: Set-UID/Set-GID decision flowchart

- exec (path, ...)
  - File referred by path has Set-UID?
  - Yes
    - ID = path owner
    - Change the process effective UID to ID of path owner
  - No
    - Do nothing
  - File referred by path has Set-GID?
  - Yes
    - ID = path GID
    - Change the process GIDs to ID only
  - No
    - Do nothing

Capabilities

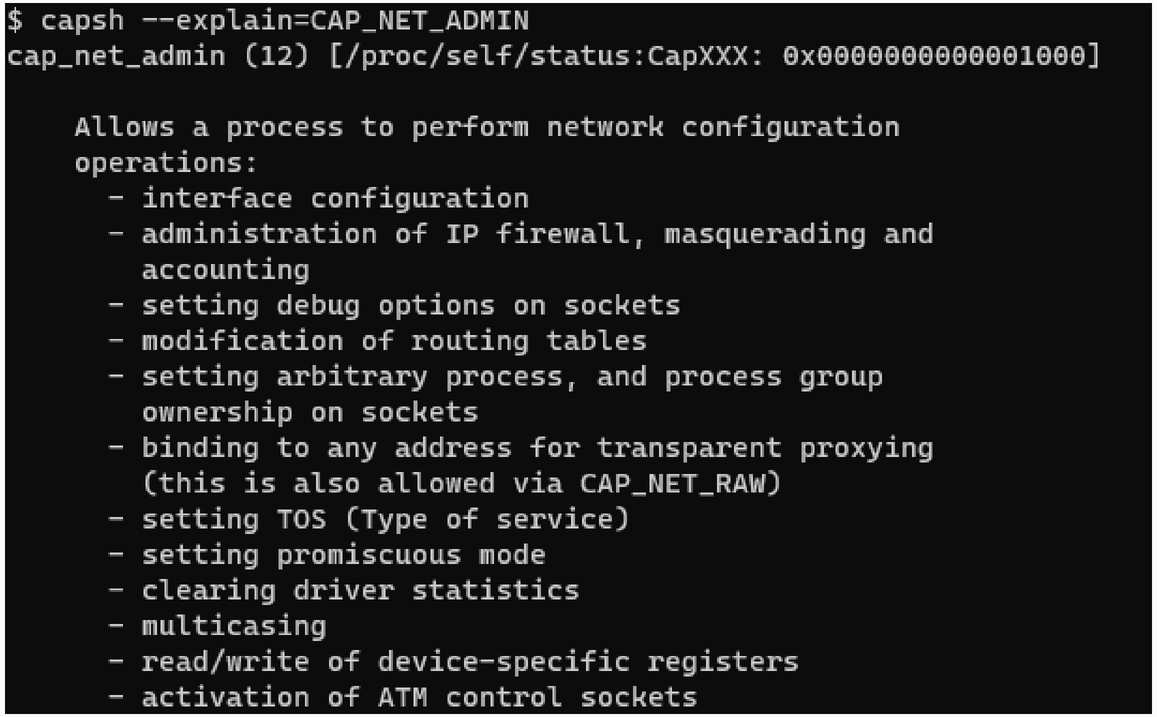
- Protection mechanism introduced in Kernel 2.2

- Allow to divide the traditional super-user privileges into distinct units
  - That can be independently enabled and disabled

- Capabilities are a per-thread attribute
  - Propagated through forks
  - Changed explicitly of by execs

List of capabilities:
Examples (small sample ...)

- ▷ CAP_CHOWN
  - Make arbitrary changes to file UIDs and GIDs
- ▷ CAP_DAC_OVERRIDE / CAP_DAC_READ_SEARCH
  - Bypass file permission / directory transversal checks
- ▷ CAP_KILL
  - Bypass permission checks for sending signals
- ▷ CAP_NET_ADMIN
  - Perform various network-related operations
- ▷ CAP_SYS_ADMIN
  - Overloaded general-purpose administration capability



```
$ capsh --explain=CAP_NET_ADMIN
cap_net_admin (12) [/proc/self/status:CapXXX: 0x0000000000001000]

    Allows a process to perform network configuration
    operations:
      - interface configuration
      - administration of IP firewall, masquerading and
        accounting
      - setting debug options on sockets
      - modification of routing tables
      - setting arbitrary process, and process group
        ownership on sockets
      - binding to any address for transparent proxying
        (this is also allowed via CAP_NET_RAW)
      - setting TOS (Type of service)
      - setting promiscuous mode
      - clearing driver statistics
      - multicasing
      - read/write of device-specific registers
      - activation of ATM control sockets
```

Capability management

- Per-thread capabilities
  - They define the privileges of the thread
  - Divided in **sets**

- Sets
  - Effective
  - Inheritable
  - Permitted
  - Bounding
  - Ambient

Thread capability sets:
Effective

- Set of capabilities used by the kernel to perform permission checks for the thread

- That is: these are the effective capabilities being used

Thread capability sets: Inheritable

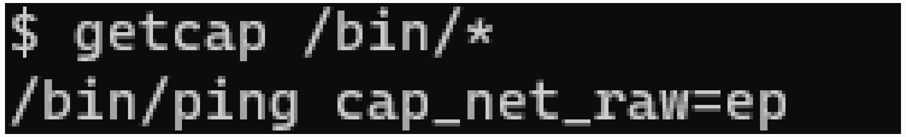
- Set of capabilities preserved across an exec
  - Remain inheritable for any program

- Are added to the permitted set when executing a program that has the corresponding bits set in the file inheritable set

Thread capability sets:
Permitted

- ▷ Limiting superset
  - For the effective capabilities that the thread may assume
  - For the capabilities that may be added to the inheritable set
    - Except for threads w/ CAP_SETPCAP in their effective set

- ▷ Once dropped, it can never be reacquired
  - Except upon executing a file with special capabilities



```
$ getcap /bin/*
/bin/ping cap_net_raw=ep
```

Thread capability sets: Bounding

- Set used to limit the capabilities that are gained during an exec
  - From a file with capabilities set
- Was previously a system-wide attribute
  - Now is a per-thread attribute

Thread capability sets:
Ambient

- Set of capabilities that are preserved across an exec of an unprivileged program
  - No set-UID or set-GID
  - No capabilities set

- Executing a privileged program will clear the ambient set

Thread capability sets: Ambient

- Ambient capabilities must be both permitted and inheritable
  - One cannot preserve something one cannot have
  - One cannot preserve something one cannot inherit
  - Automatically lowered if either of the corresponding permitted or inheritable capabilities is lowered

- Ambient capabilities are added to the permitted set and assigned to the effective set upon an exec

Files extended attributes (xattr)

- Files' metadata in UNIX-like systems
  - Some not interpreted by kernels
- Linux: key-value pairs
  - Keys can be defined or undefined
  - If defined, their value can be empty or not
  - Key's namespaces
    - `namespace.attr_name[.attr_name]`

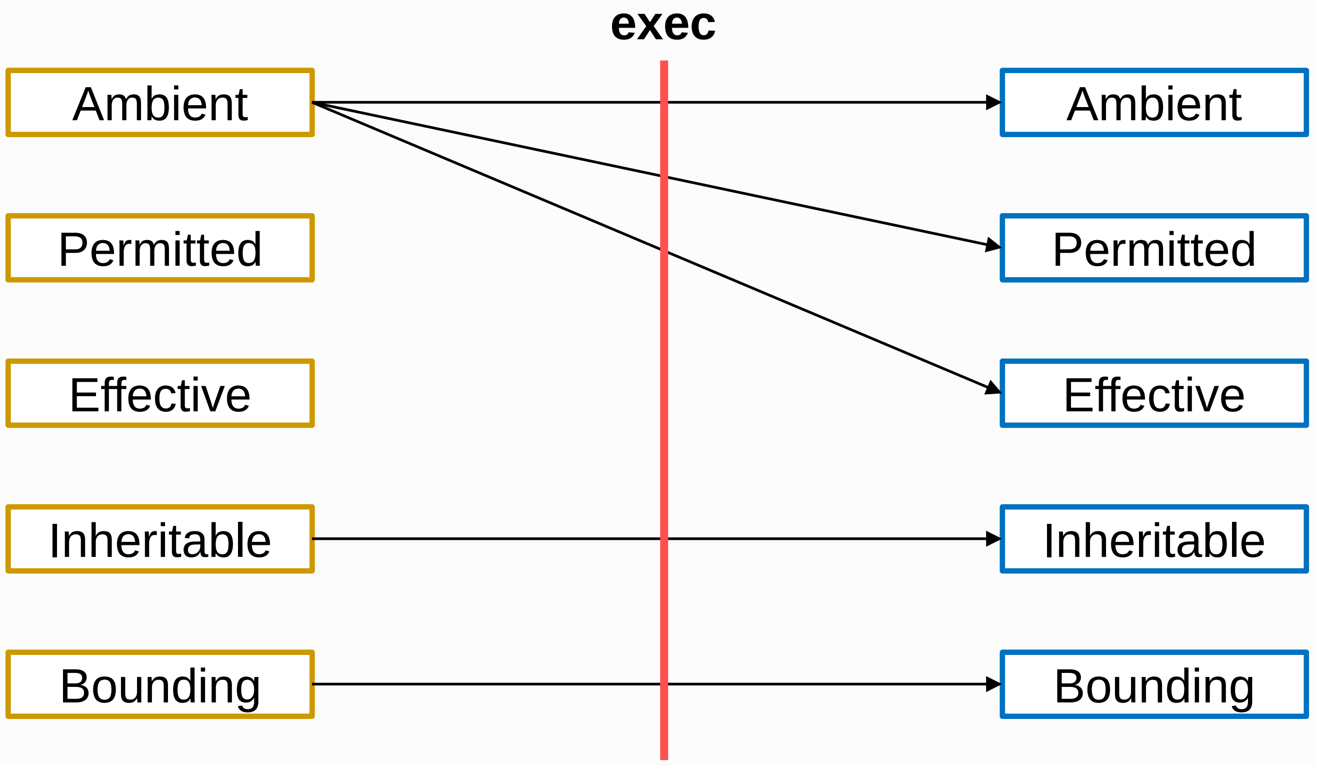
- Namespaces
  - security
    - For files' capabilities
    - setcap / getcap
  - system
    - ACL
  - trusted
    - Protected metadata
  - user
    - setfattr / lsattr / getfattr

File capabilities

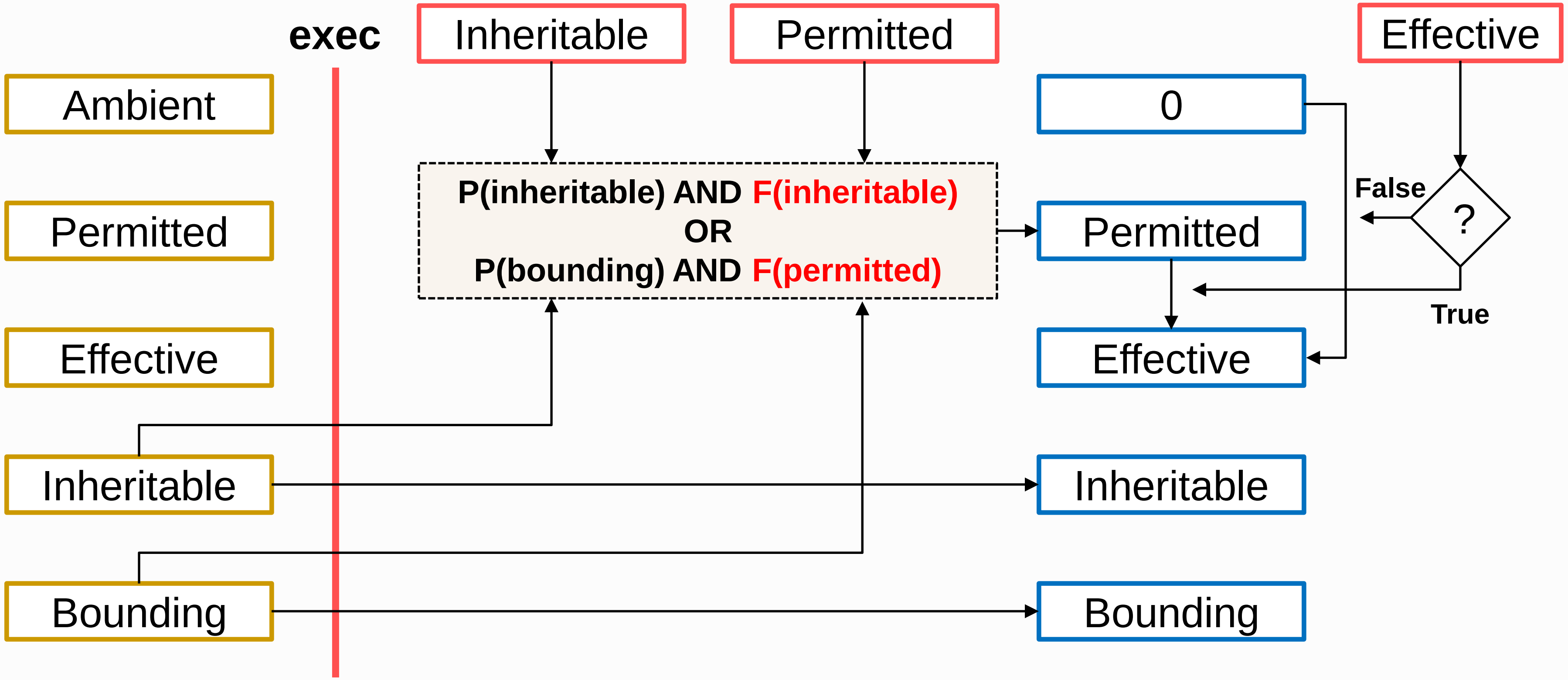
- Stored in the security.capability attribute

- Specify capabilities for threads that exec a file
  - Permitted set
    - Immediately forced into the permitted set
    - Previous AND with the thread’s bounding set
  - Inheritable set
    - To AND with the threads’ inheritable set
    - Can be used to reduce the effective set upon the exec
  - Effective bit
    - Enforce all new capabilities into the thread’s effective set

Capability transfer across exec: No privileged files



Capability transfer across exec (non-root) Privileged files



Capability transfer across exec (root)

- EUID = 0 or RUID = 0
  - Capability sets are considered to be all 1's
- EUID = 0
  - File effective bit considered 1
- Exception: EUID = 0, RUID ≠ 0
  - Set-UID file was executed
  - File capabilities are honored if present

Control groups (cgroups)

- Collection of processes bound by the same criteria and associated with a set of parameters or limits

- cgroups are organized hierarchically
  - cgroup file system
  - Limits can be defined at each hierarchical level
    - Affecting the sub-hierarchy underneath

- Subsystems
  - Kernel component that modifies the behavior of cgroup processes
  - Resource controllers (or simply controllers)

cgroups v1 and v2

- Currently two versions coexist
  - But controllers can only be used in on of them

cgroups file system

- This file system is created by mounting several controllers as cgroup-type file system entities
  - Usually /sys/fs/cgroup
  - In V2 all controllers are part of a single cgroup2

- Each controller defines a tree of cgroups below the mount point
  - e.g. memory controller → /sys/fs/cgroup/…/memory.[…]

cgroup V2 (and V1) controllers

- cpu (cpu & cpuacct in V1)
  - CPU usage & accounting
- cpuset
  - CPU bounding
- memory
  - Memory usage & accounting
- devices
  - Device creation & usage
- freezer
  - Suspend/resume groups of processes
- Io (blkio in V1)
  - Block I/O management
- perf_event
  - Performance monitoring
- hugelb
  - Huge pages management
- pids
  - # of processes in cgroup
- rdma
  - RDMA / IB resources' management
- Deprecated from V1
  - net_cls
    - Outbound packet classification
  - net_prio
    - Network interfaces priorities

cgroup V2 definition

- Directory under /sys/fs/cgroup
  - With a set of controllers defined by cgroup.controllers
  - With hierarchy limits defined by cgroup.depth and cgroup.descendants
  - With files to send KILL signals (cgroup.kill) and freeze/unfreeze orders (cgroup.freeze) to all cgroup processes
    - Including descendants
  - The processes using the cgroup are given by cgroup.procs and their status reported by cgroups.events
    - We can add a process to a cgroup just by writing its PID on the first file

- For each active controller, specific files will exist

- Processes can only belong to leaf cgroups
  - “No internal processes” rule

cgroups of a process

- A process can be controlled by an arbitrary number of cgroups

- The list of a process' cgroups is given by the /proc file system
  - /proc/[PID]/cgroup

Linux Security Modules (LSM)

- Framework to add new Mandatory Access Control (MAC) extensions to the kernel

- Those extensions are not kernel modules
  - They are embedded in the kernel code
  - They can be activated or not at boot time
  - List of extensions given by /sys/kernel/security/lsm

LSM extensions

- Capabilities (default)
- AppArmor
  - MAC for applications
- LoadPin
  - Kernel-loaded files origin enforcement
- SELinux
- Smack
  - Simplified Mandatory Access Control Kernel
- TOMOYO
  - Name-based MAC extension
- Yama
  - System-wide DAC security protections that are not handled by the core kernel itself
- SafeSetID
  - Restricts UID/GID transitions

Source: https://www.kernel.org/doc/html/next/admin-guide/LSM/index.html

AppArmor

- Enables the definition of per-application MAC policies
  - Profiles
  - Applications are identified by their path
    - Instead of i-node
- Profiles restrict applications' actions to the required set
  - All other actions will be denied
- Profiles define
  - Actions white-listed
  - Logging actions

AppArmor: profiles

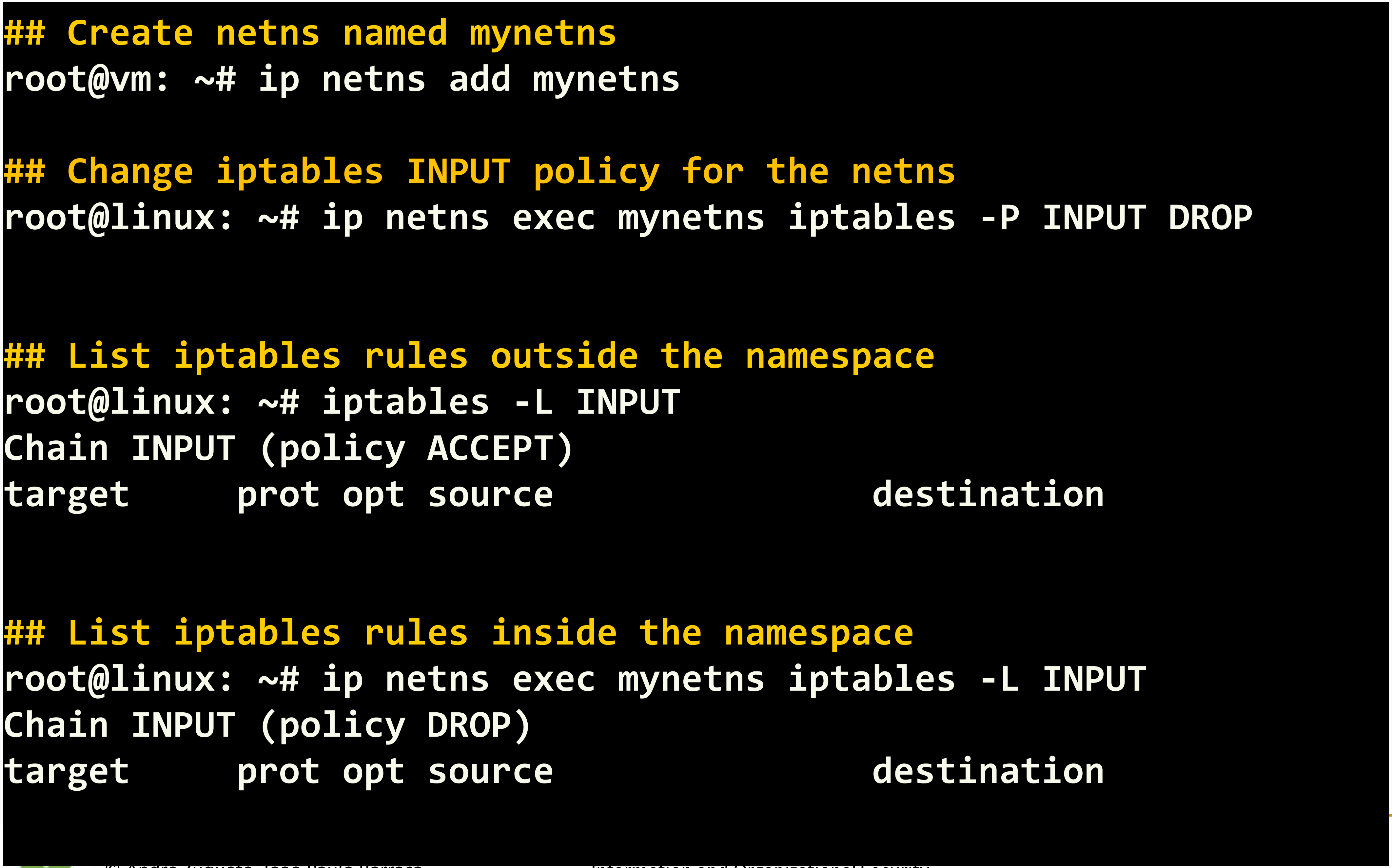
- Profiles are loaded into the kernel
  - Upon compilation from textual files
  - apparmor_parser

- Profiles can be used on a voluntary basis
  - aa-exec

Confinement: Namespaces

- Allows partitioning of resources in views (namespaces)
  - Processes in a namespace have a restricted view of the system
  - Activated through syscalls by a simple process:
    - clone: Defines a namespace to migrate the process to
    - unshare: disassociates the process from its current context
    - setns: puts the process in a Namespace

- Types of Namespaces
  - **Mount**: Applied to mount points
  - **process id**: first process has id 1
  - **network**: "independent" network stack (routes, interfaces...)
  - **IPC**: methods of communication between processes
  - **uts**: name independence (DNS)
  - **user id**: segregation of permissions
  - **cgroup**: limitation of resources used (memory, cpu...)



```
## Create netns named mynetns
root@vm: ~# ip netns add mynetns

## Change iptables INPUT policy for the netns
root@linux: ~# ip netns exec mynetns iptables -P INPUT DROP


## List iptables rules outside the namespace
root@linux: ~# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination


## List iptables rules inside the namespace
root@linux: ~# ip netns exec mynetns iptables -L INPUT
Chain INPUT (policy DROP)
target     prot opt source               destination
```

```
## List Interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00


## Move the interface enp0s3 to the namespace
root@linux: ~# ip link set enp0s3 netns mynetns

## List interfaces in the namespace
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT...
    link/ether 08:00:27:83:0a:55 brd ff:ff:ff:ff:ff:ff


## List interfaces outside the namespace
root@linux: ~# ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Confinement: Containers

- ▷ Explores namespaces to provide a virtual view of the system
  - Network isolation, cgroups, user ids, mounts, etc…
- ▷ Processes are executed under a container
  - Container is an applicational construction and not of the core
  - Consists of an environment by composition of namespaces
  - Requires building bridges with the real system network interfaces, proxy processes
- ▷ Relevant approaches
  - **LinuX Containers**: focus on a complete virtualized environment
    - evolution of OpenVZ
  - **Docker**: focus on running isolated applications based on a portable packet between systems
    - uses LXC