APISEC interactive workshop:
Application-level access control
for API-based cloud applications

Architecture, tactics, patterns and technologies

Bert Lagaisse, 2021/03/25

Bert.lagaisse@kuleuven.be

KU LEUVEN DistriNet

Overall 3-phase approach

Application-level access control for API-based cloud applications

Application-driven requirements analysis

Example case studies
Example architectures and functional decompositions
Example security requirements and their variations
Feedback and refinement based on your case studies

Possible architectural solutions and their trade-offs

Security architecture: tactics, solutions and trade-offs
Their support in OAUTH and IdM systems.
Support in actual technologies and implementations

Advanced server-side access control

Overview of server-side access control models
ABAC, PBAC and multi-tenancy support
Secure data access: crypto or policy ?

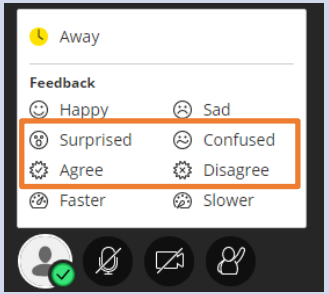
The goal of today: Patterns, tactics, building blocks

Common problems and their possible solutions

Reference Architecture

- Basic assumptions
- API Consumer types
  - Client-focused flows
  - Server-focused flows
- Tenant-side daemons
  - The good, the bad & the ugly
- Server-side daemons
  - “united we stand”

Brain storm and discussion moments:



Solution mapping

- Consumer type – flow
- Flow – token type
- Flow – technology
  - Keycloak
  - Azure AD
  - Cognito
  - …
- Going beyond state of practice

Brainstorm:
- Give us feedback based on your case studies
- Do you recognize your system in (part of) this architectural solution
- Gradual refinement based on your feedback

Quick recap:
2 example case studies
Reference architecture overview

2 example case studies

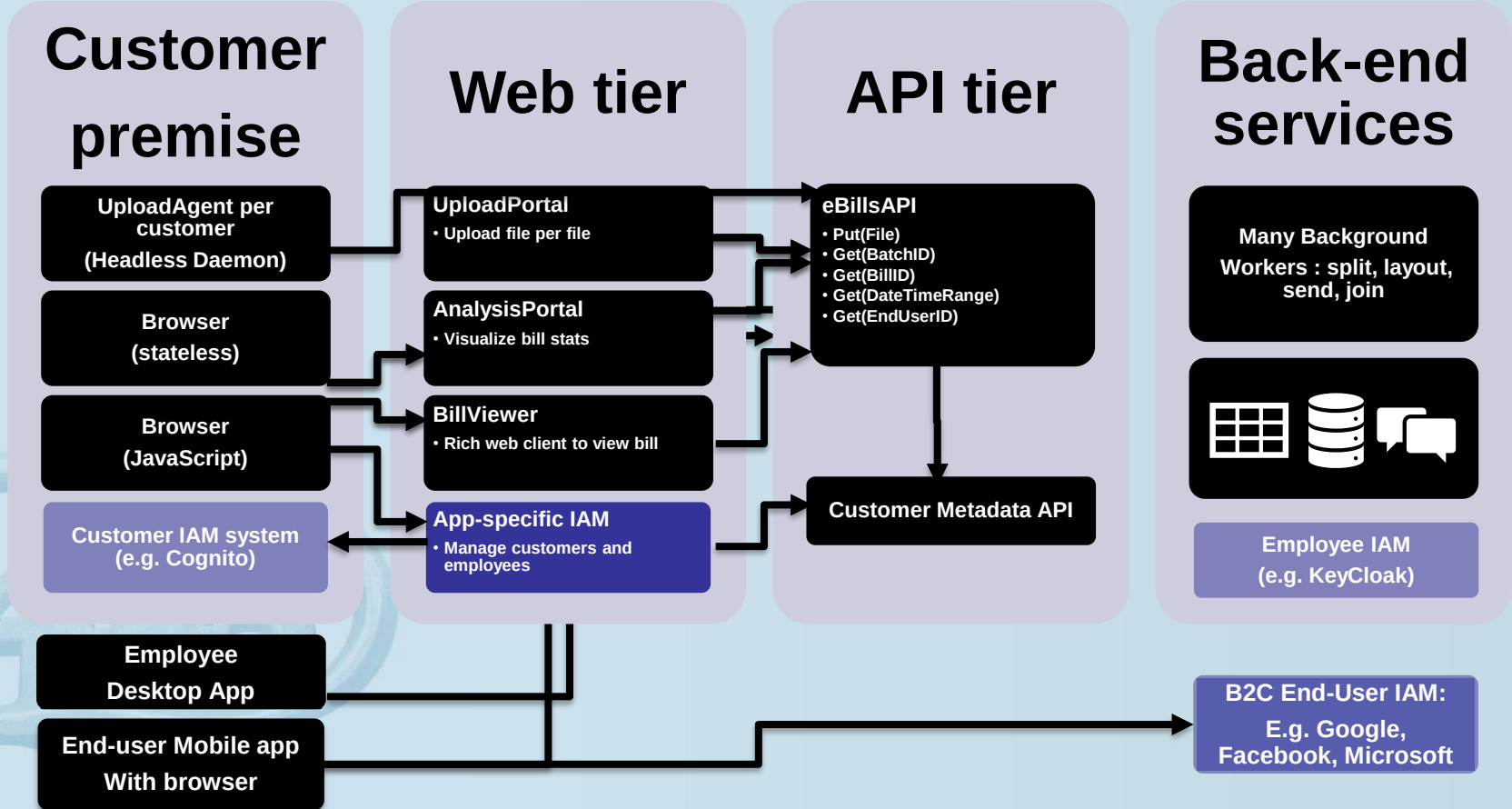
E-Invoice

- E.g.: telecom provider sends out batch of invoices to customers
  - E-invoice offers this as-a-service
- Use cases:
  - Telecom provider uploads raw invoicing data to eInvoice
    - Via employee on a web interface
    - Via a batch upload daemon
  - eInvoice does templating and sends out the invoices via, e-mail, snail-mail or zoomit
    - Telecom and E-invoice employees can configure templates, edit batches, etc…
  - End-user can inspect invoice online
    - End-user = customer of the customer.

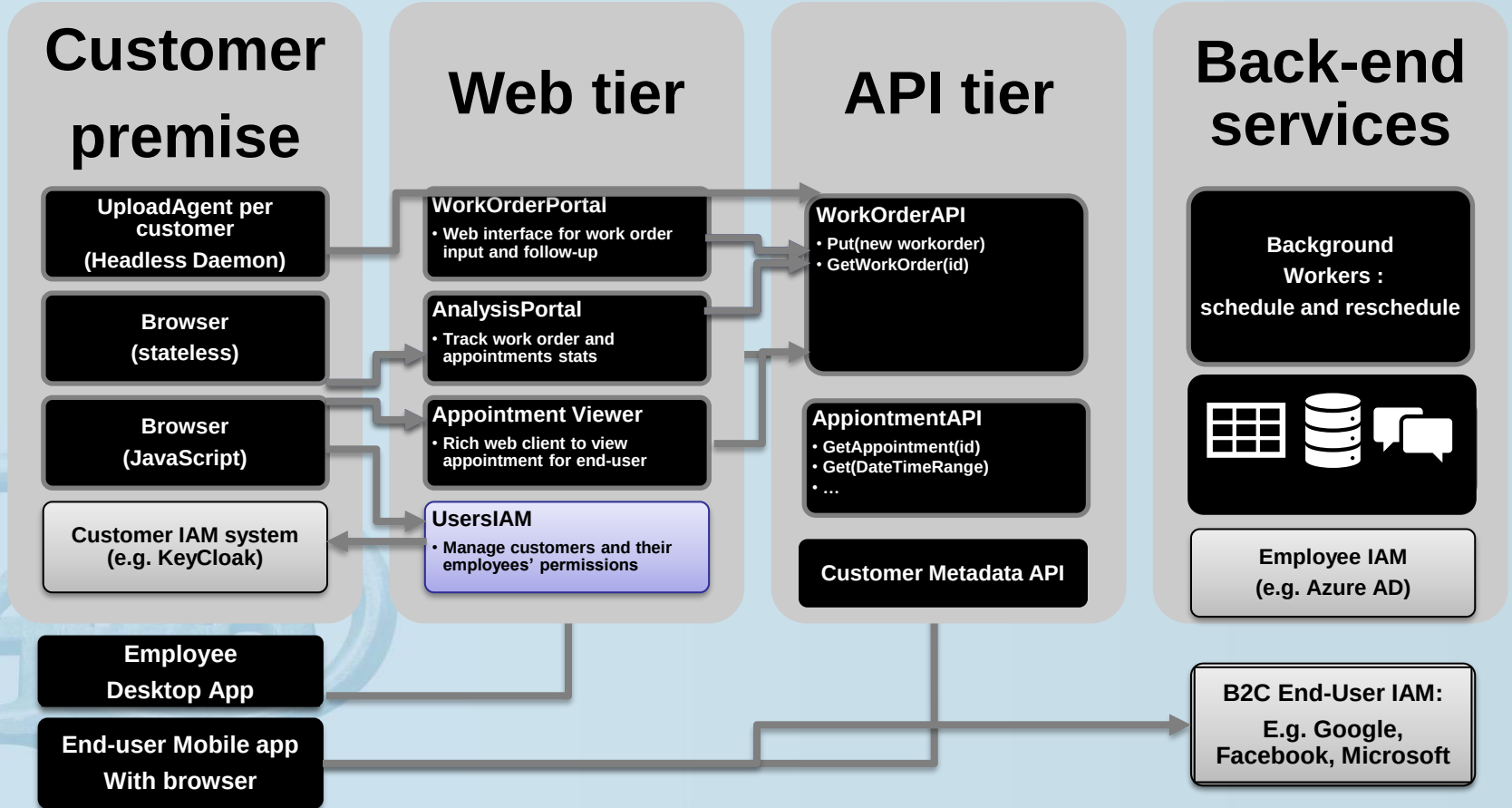
e-Workforce

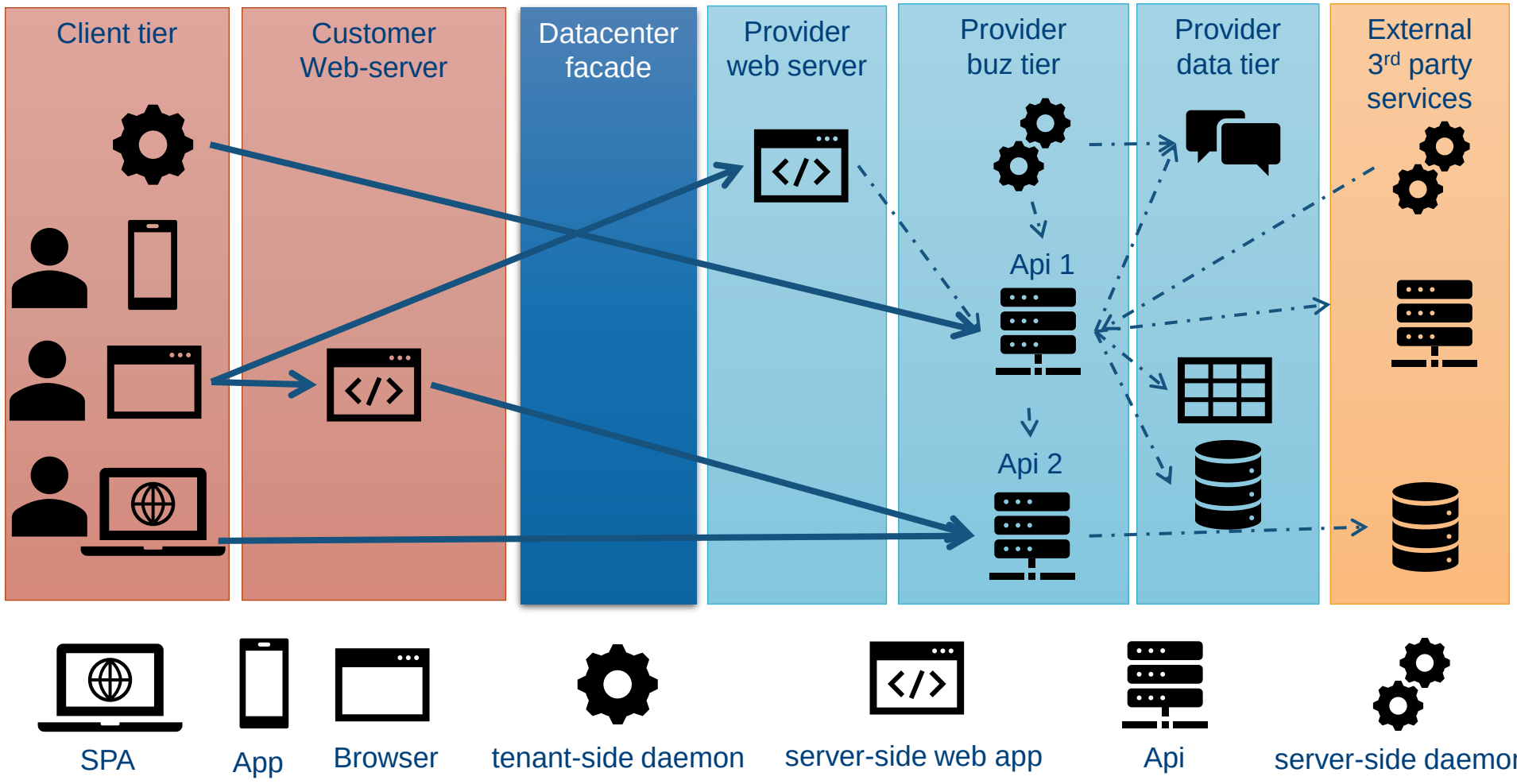
- Telecom provider uses many technicians to install network devices in customer's premisses
  - eWorkforce provides technicians and schedules appointments at customer
- Use cases:
  - Telecom provider uploads batch of workorders at its customers
  - eWorkforce schedules the appointments and plans technicians
  - End-user can see appointment
  - Technician can report used materials for billing

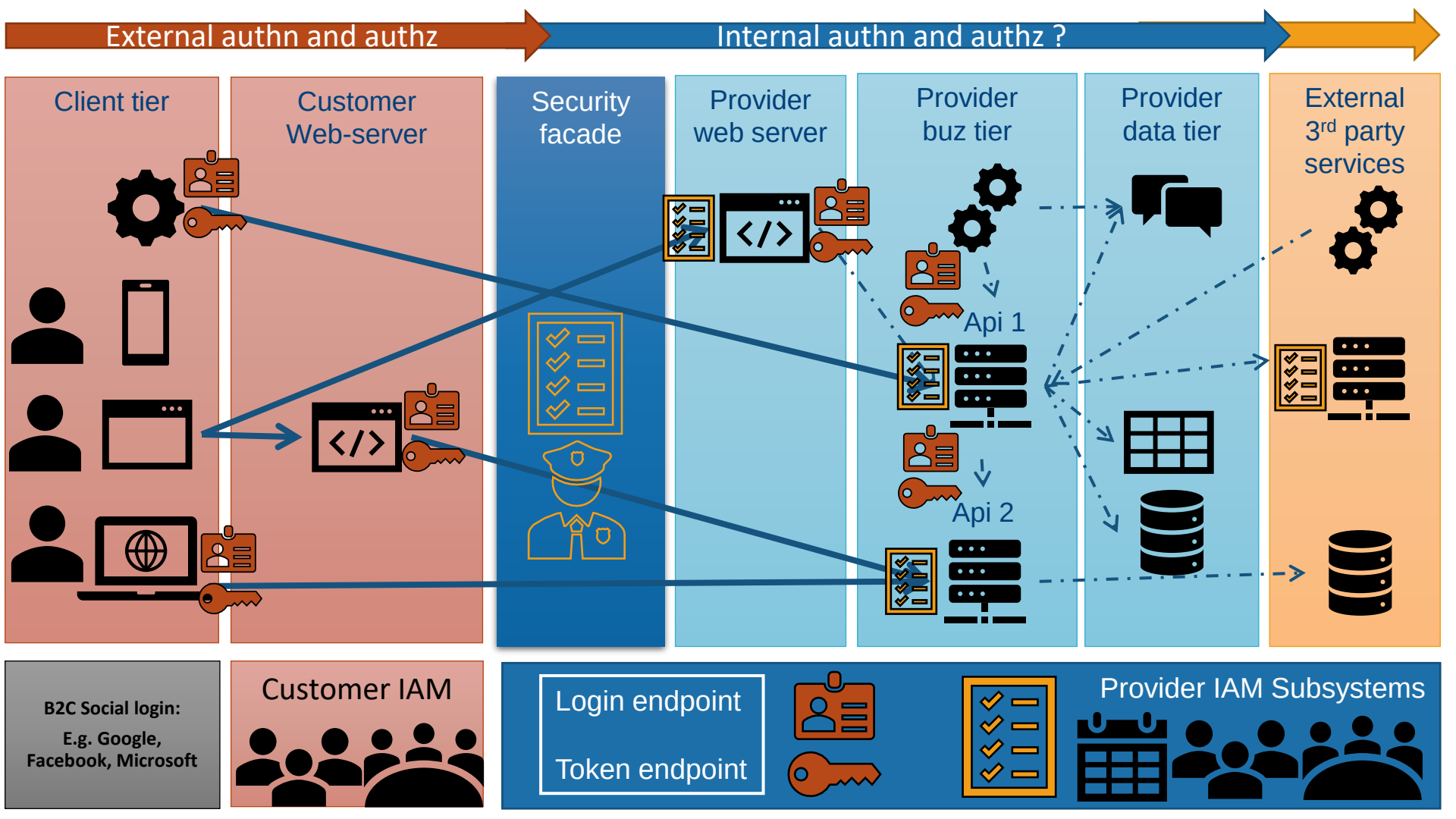
Basic (m)architecture of eBills application
Functional subsystems and logical decomposition



Basic (m)architecture of eWorkForce application
Functional subsystems and logical decomposition




Client tier
Customer Web-server
Datacenter facade
Provider web server
Provider buz tier
Provider data tier
External 3rd party services
Api 1
Api 2
SPA
App
Browser
tenant-side daemon
server-side web app
Api
server-side daemon


External authn and authz
Internal authn and authz ?
Client tier
Customer Web-server
Security facade
Provider web server
Provider buz tier
Provider data tier
External 3rd party services
Api 1
Api 2
B2C Social login:
E.g. Google, Facebook, Microsoft
Customer IAM
Login endpoint
Token endpoint
Provider IAM Subsystems

Feedback from the previous workshop

Previous meetings vs today's meeting

- Oauth basics
- Application cases
  - Variating subsystems
  - Variating consumers of API
- Reference architecture
  - types of sub systems
  - Deployment view
- Analysis, requirements, concerns
  - Wrt identity and access management
  - Wrt authentication and authorization

- Solutions, architecture, tactics, patterns, …
  - Multi-tenancy and IaM
  - Advanced token acquisition
    - For authn/authz
    - Client-focused
    - Server-focused
  - Fighting your daemons
    - Tenant-side
    - Provider-side

Feedback from previous workshop

Current approaches and interests

- Current approaches
  - Rest-based APIs
  - Externalized IAM
    - Self-managed (e.g. KeyCloak)
    - As a service (e.g. Azure AD)
  - OpenID Connect and Oauth
  - Identity and access tokens:
    - Jwt and claims
- Major interests
  - Daemons
    - On client
    - On server-side
  - Token flows with a client-focus
    - Sender-constrained tokens
  - Token flows with a server-focus
  - Identity delegation/impersonation
    - for distributed flows
  - Secret management
    - For calling external services

Reference Architecture & architectural solutions: basics and focus

Authentication, authorization

Authentication (AuthN)

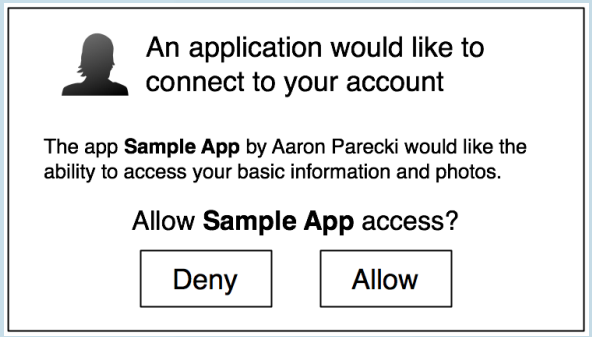
- The process of proving that someone or something is who she/he/it claims to be.

- E.g using OpenID Connect
- Results in a proven identity

Authorization (AuthZ)

- Grant an authenticated entity(someone or something) permission to do something

- E.g. using the OAuth 2.0 protocol
- Results in an access token

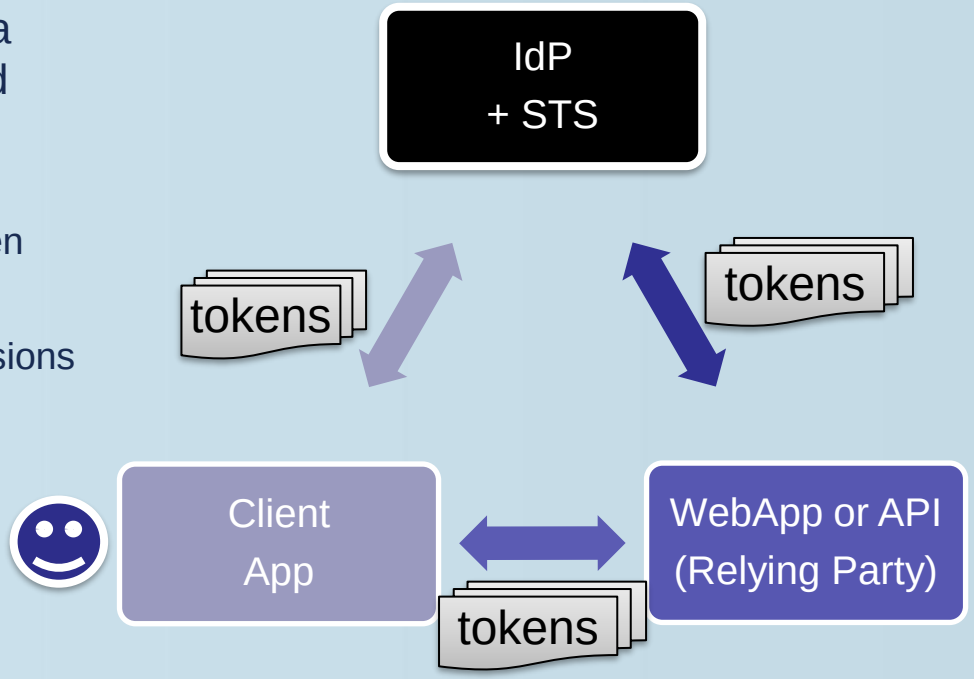


Externalizing identity and access management: Based on security tokens

Security tokens

- **Identity token:** proves identity of a user as verified by the externalized identity provider (IdP)
  - **Security token server** (STS)
  - E.g. SAML or OpenID Connect token
- **Access token: Oauth 2.0**
  - Contains app id (client) and permissions to access a resource (audience)
  - Often shorter-lived TTL
- **Refresh token:**
  - Longer-lived
  - To get a new access token

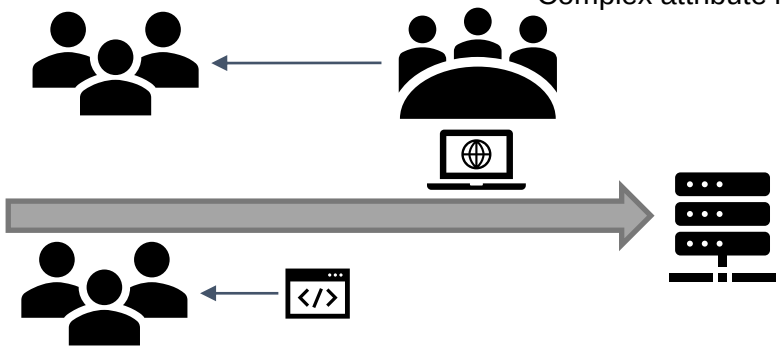
Parties



Single-tenant vs Multi-tenant API: token validation

How to support multi-tenancy ?

- Single Tenant API with multitenant IAM:
  - API bound to 1 IAM system of the provider
    - API only trusts this 1 IAM system
    - Not considering dedicated tenant deployments
  - Token validation library handles most checks
    - Application-level validation of permissions
  - Provider IAM system can delegate and redirect to tenant IAM for user authn and authz
    - Provider IAM translates attributes in tenant tokens to provider attributes.
- Multi-tenant API
  - Token validation library: Any token coming from any IAM system accepted
    - OR Multi-plex validation config per tenant…
  - Application-level logic:
    - Complex validation if token is coming from an “accepted tenant IAM system”.
    - Complex validation if token is coming from an accepted client application.
    - Complex attribute mapping



KU LEUVEN DistriNet

Token acquisition from several types of apps:

consumer tier

SPA

server-side web app

Mobile App

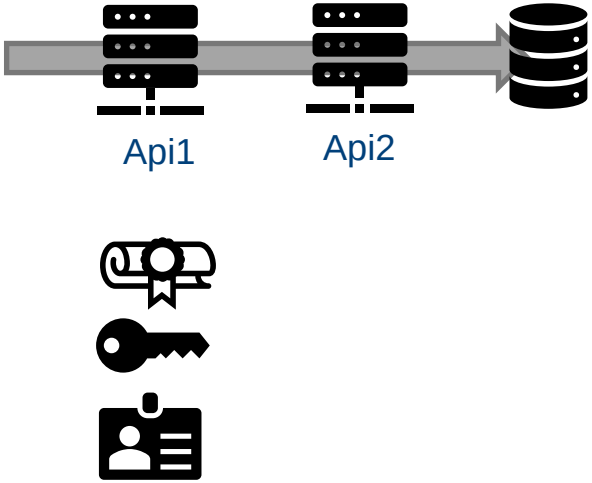
Browserless app

Desktop app

IOT device

tenant-side daemon

› With a signed-in-user



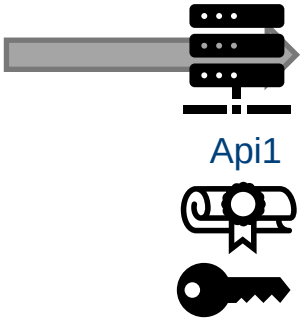
consumer tier

› As application

server-side daemon

tenant-side daemon

daemon web app



Token acquisition: API Consumer-provider variation scenarios

Supporting client variation with OAuth 2.0 and OpenID connect

OAuth 2.0:

Authorization protocol

- Variation per type of client: basics
  - Implicit grant flow
  - Authorization code grant (+PKCE)
  - Resource owner password credentials grant (ROPC)
  - Client credentials grant
- Flows beyond basics/standards:
  - Hybrid flow
  - On-behalf-of flow
    - Token-exchange flow
  - Device code flow

OpenID Connect:

Authentication protocol

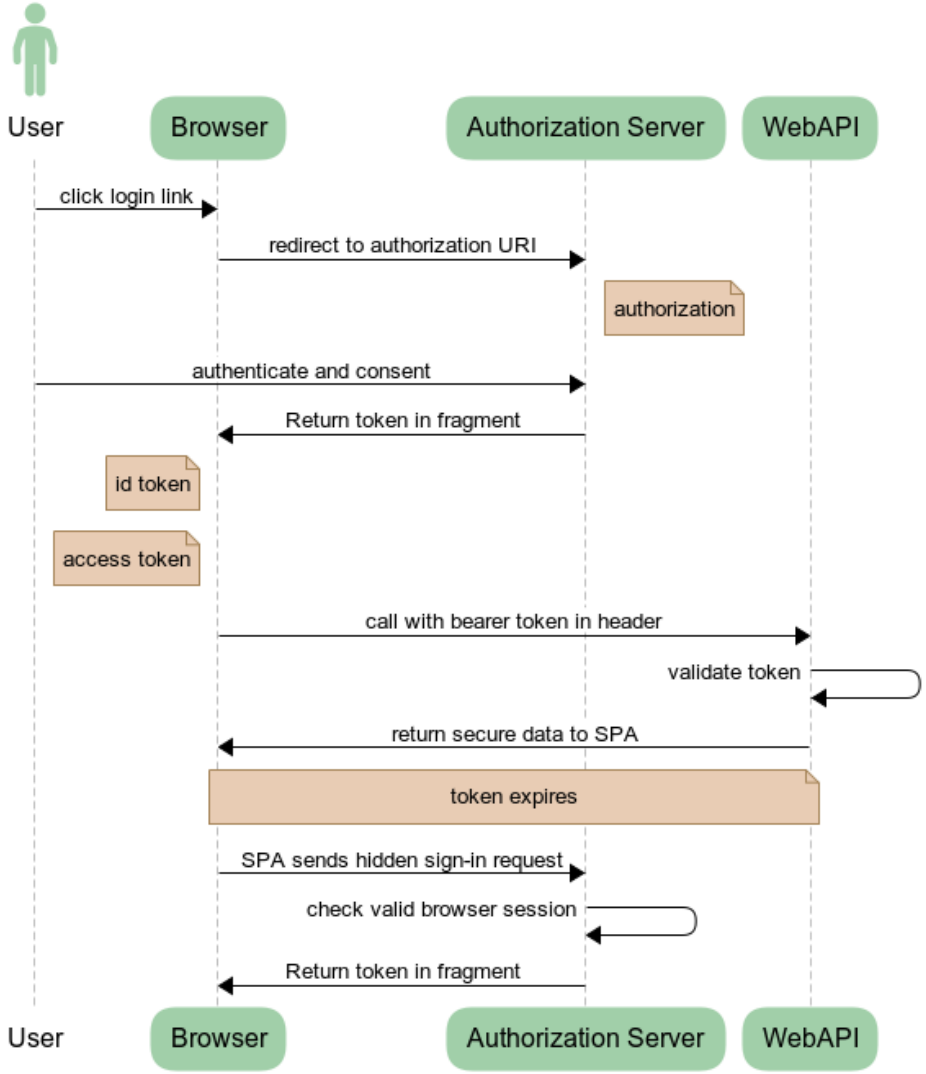
- Securely login a user to an application
- Interactive authentication
- Without exposing password to app
- Built on OAuth 2.0
- Extends OAuth 2.0 for use as an authentication protocol

Token acquisition: client-side token-exposure and storage

Single page app

Token acquisition variation

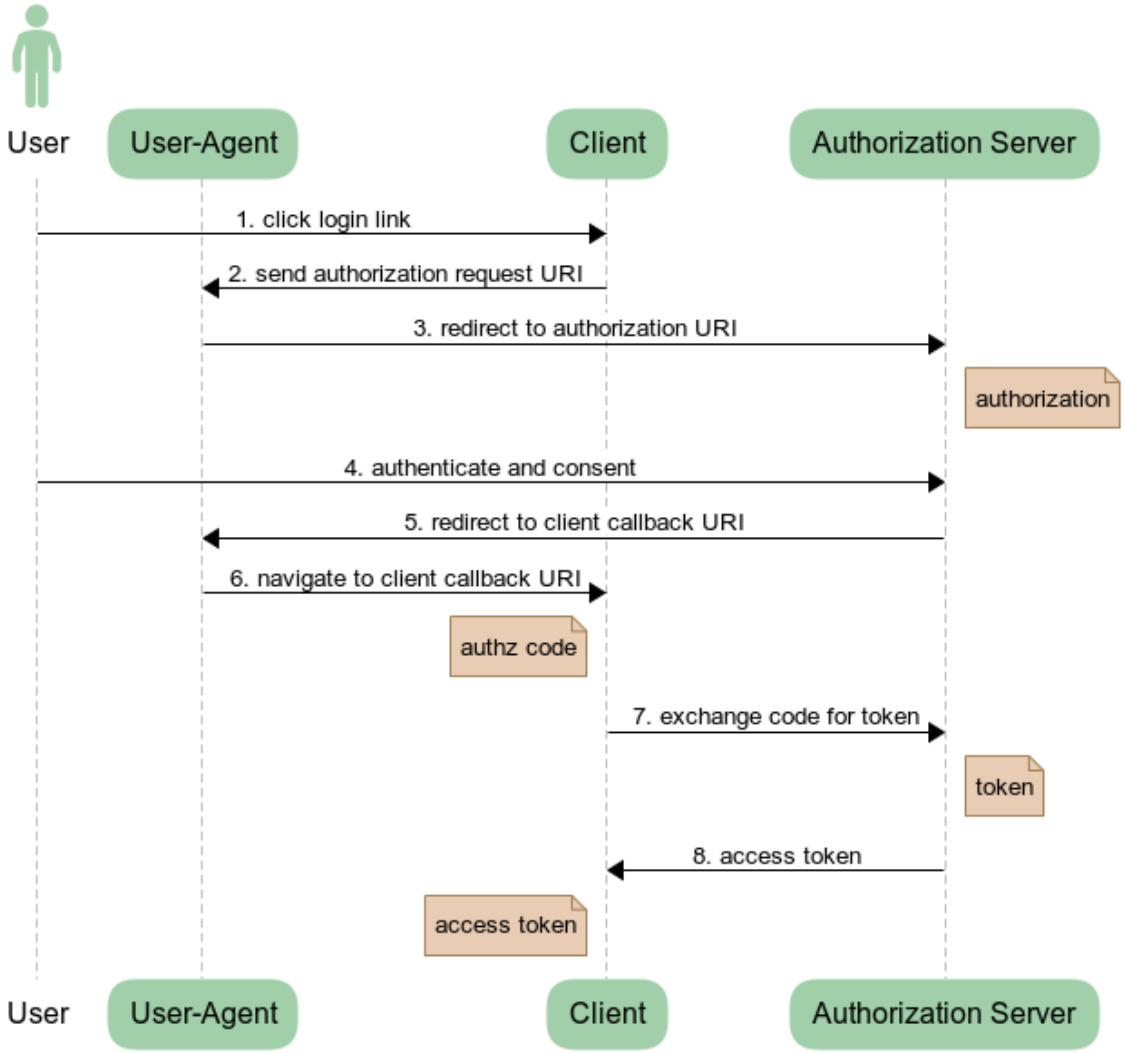
- Implicit flow
  - Get ID token and access token directly from authorization endpoint
    - In url fragment !
    - No refresh token
- Authorization code flow
  - First get a “short” authorization code
  - Get access token with this code
  - Refresh token to renew access token
- Hybrid flow
  - ID token + authz code from authz endpoint instead of token endpoint
  - Access token from token endpoint
- PKCE ?



Authz code grant flow

e.g. client app with browser

- url fragment might be too limited for
  - All claims
  - All access tokens
  - Protecting the token
- Authorization code flow
  - First get a “short” authorization code
  - Get access token with this code
  - Refresh token to renew access token
- Refresh token must be protected!
  - Should be rotating!
  - Should be sender-constrained!

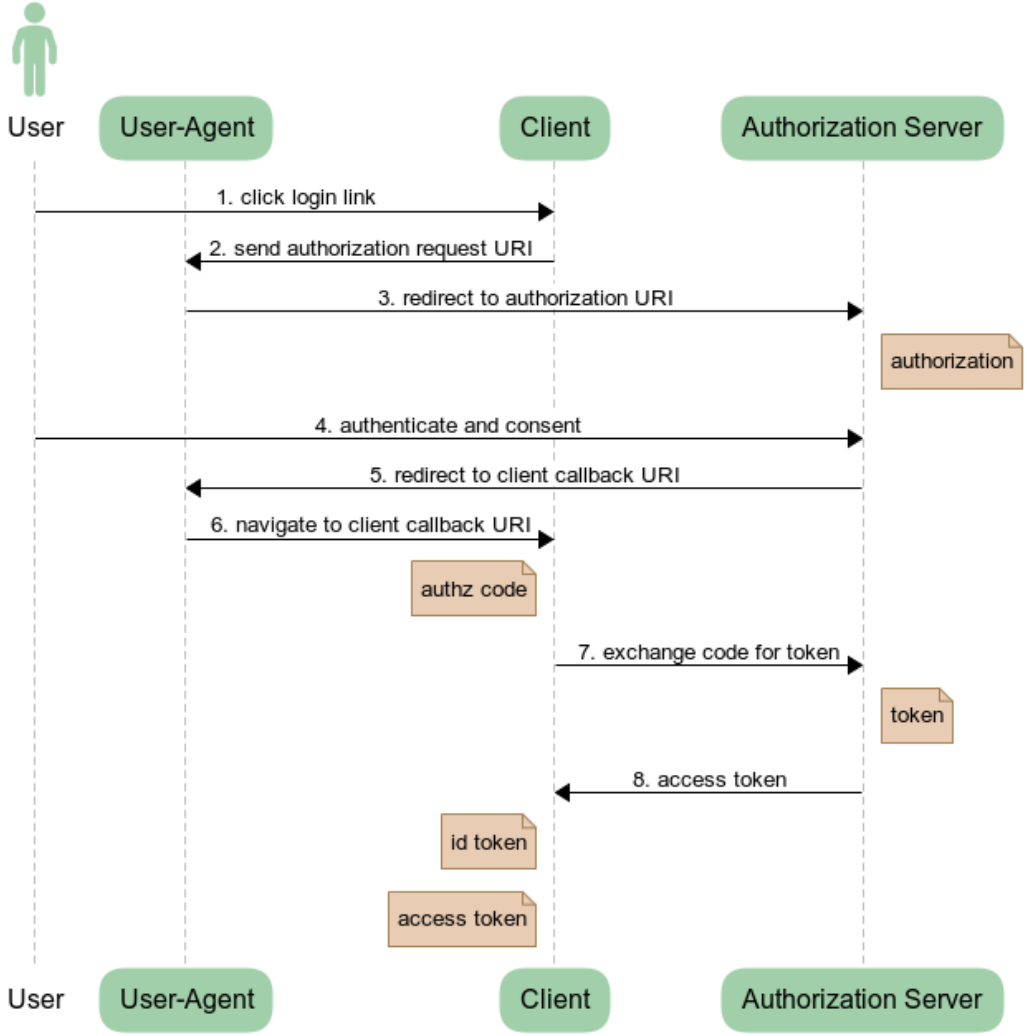


Auth code grant flow (2)

Id token also via code

- Both access token
  and id token

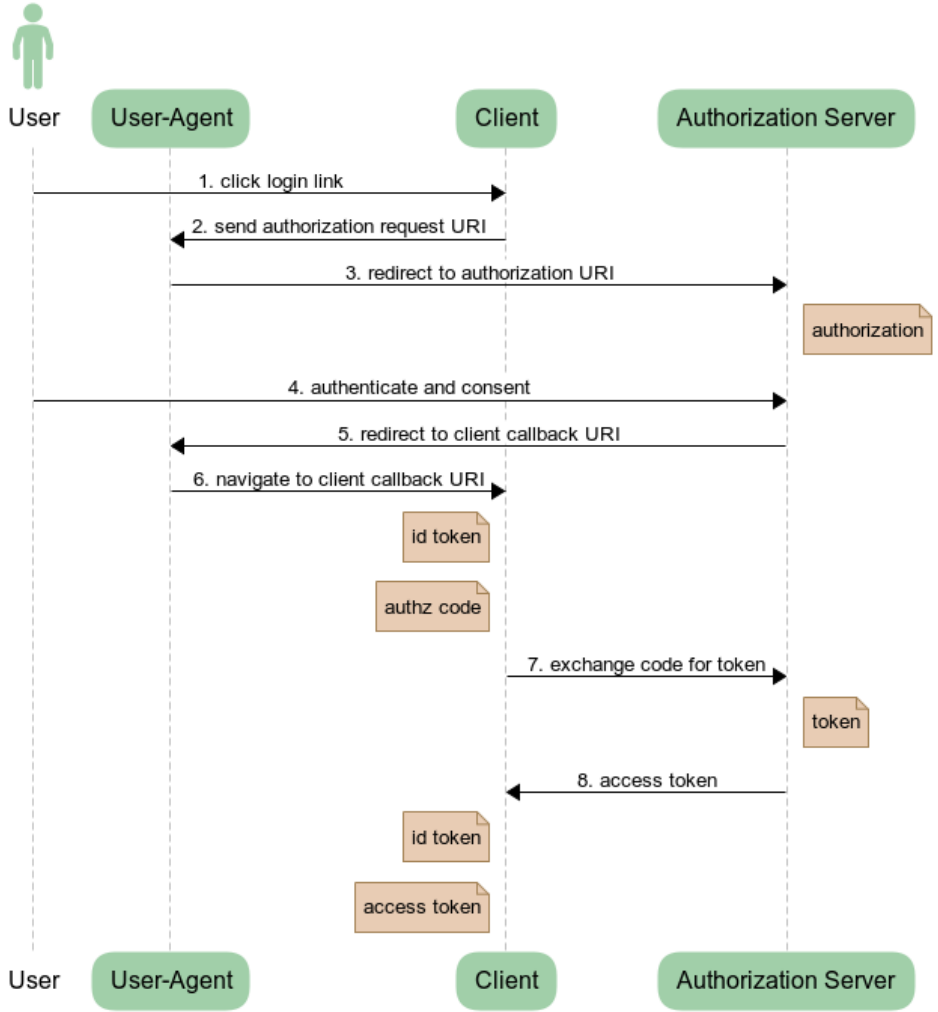
  are obtained via authz code



Hybrid flow

Mix it. Fix it?

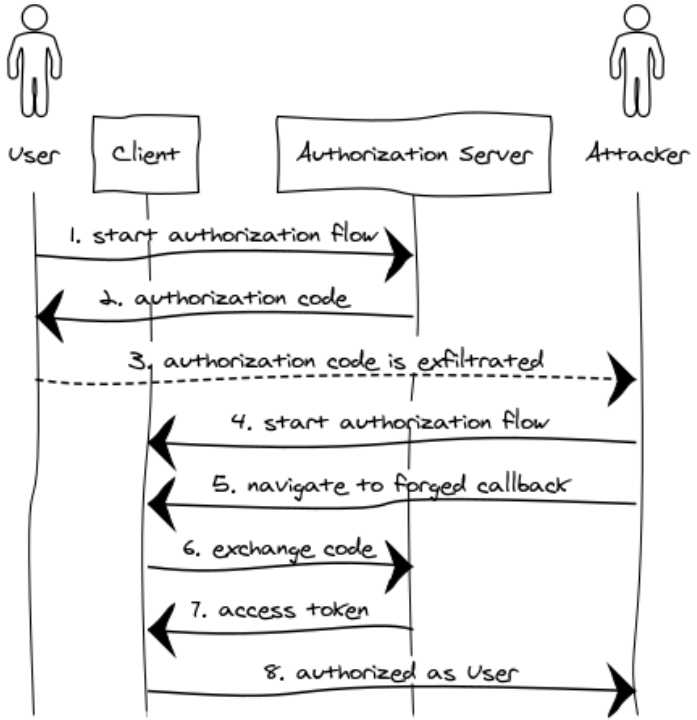
- ID token + authz code from authz endpoint
  - Access token from token endpoint
- Quote from Scott Brady (IdentityServer4):
  - Exposes tokens to the front-end
    - in the same way as the implicit flow.
    - acceptable when only dealing with identity tokens,
    - assuming you are using nonce validation.
  - Requesting an authorization code and an identity token from the authorization endpoint at the same time can even be advantageous for the client application.
    - Performing nonce validation early can be beneficial,
    - Checking the code hash (c_hash) allows the client application to detect authorization code injection.



Security considerations: implicit, authz code, hybrid

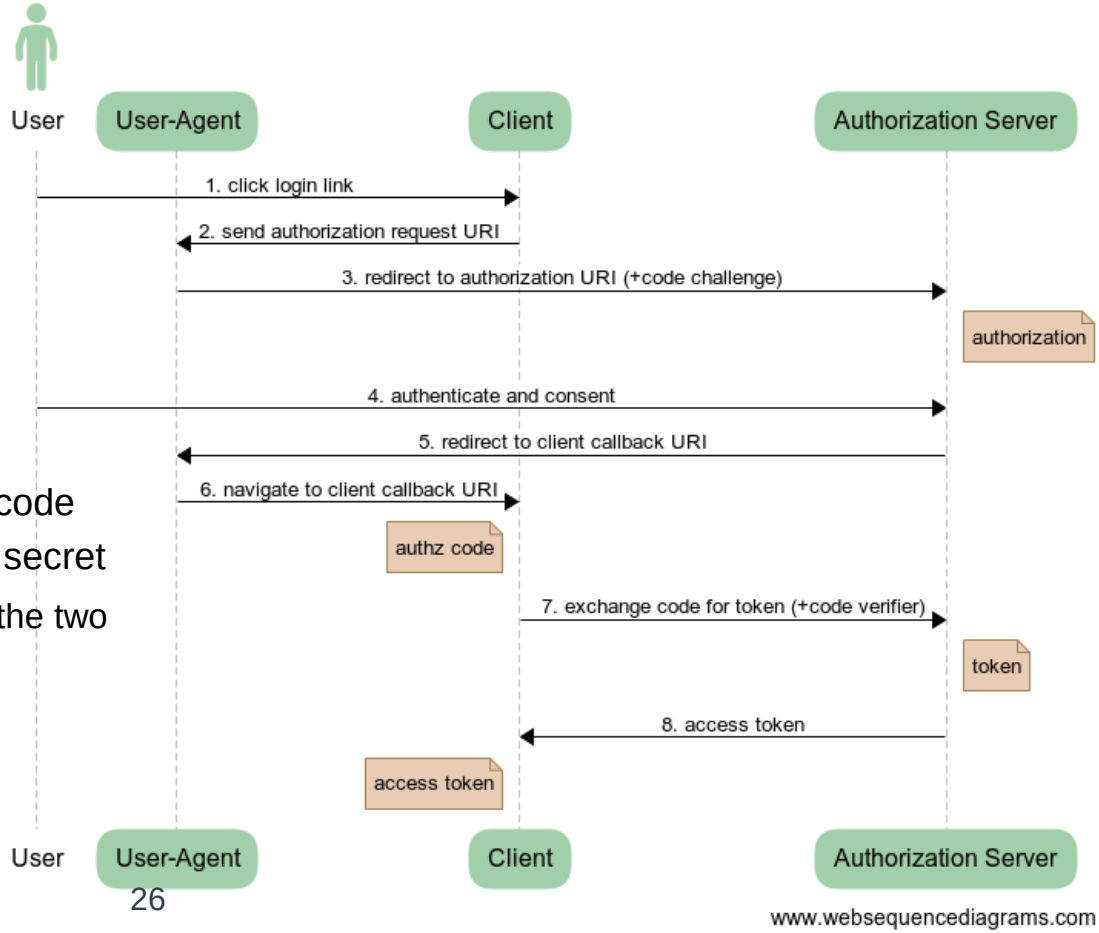
PUBLIC client-side consumers: SPA, Desktop and Mobile apps

- Client receives the authorization code from the redirect URI.
  - App parses redirect URI in browser
  - protect against other apps on device
- Avoid Implicit flow grant
  - Easier, but can't offer this protection
  - Other apps can subscribe for redirect URI and steal the access token
- use PKCE: Proof Key for Code Exchange
- Weak spot: refresh token !



Proof Key for Code Exchange (PKCE)

- Bind an authorization code to a client's session
  - Client generates a random secret per authorization request
  - Client sends the hashed secret in the authorization request
  - When it exchanges the authorization code for an access token, it also sends the secret
    - The server can hash and compare the two hashes



Proof Key for Code Exchange (PKCE)

REQUEST

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
    &code_challenge=rLGaLy…5Z5Dc&code_challenge_method=S256 HTTP/1.1
Host: server.example.com
```

REQUEST

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&code_verifier=8WBGM8cbVT…bRzqts370
```

Other techniques for sender-constrained tokens

Next to PKCE

- Demonstration of Proof-of-Possession
  - Client presents public key and access token to the Authorization Server
    - Encoded as JWT
    - Signed with private key of client
  - API can verify that client possesses the private key of same keypair
- Mutual TLS Sender Constrained access token
  - Bound to underlying mutual TLS connection between client and AS
- Both require asymmetric key pair management or PKI/Certificates

Refresh token rotation

Authorization servers must rotate refresh token on each use

- on each request to exchange a refresh token for a new access token
  - Return new refresh token
  - Invalidate previous refresh token
- Especially for public clients

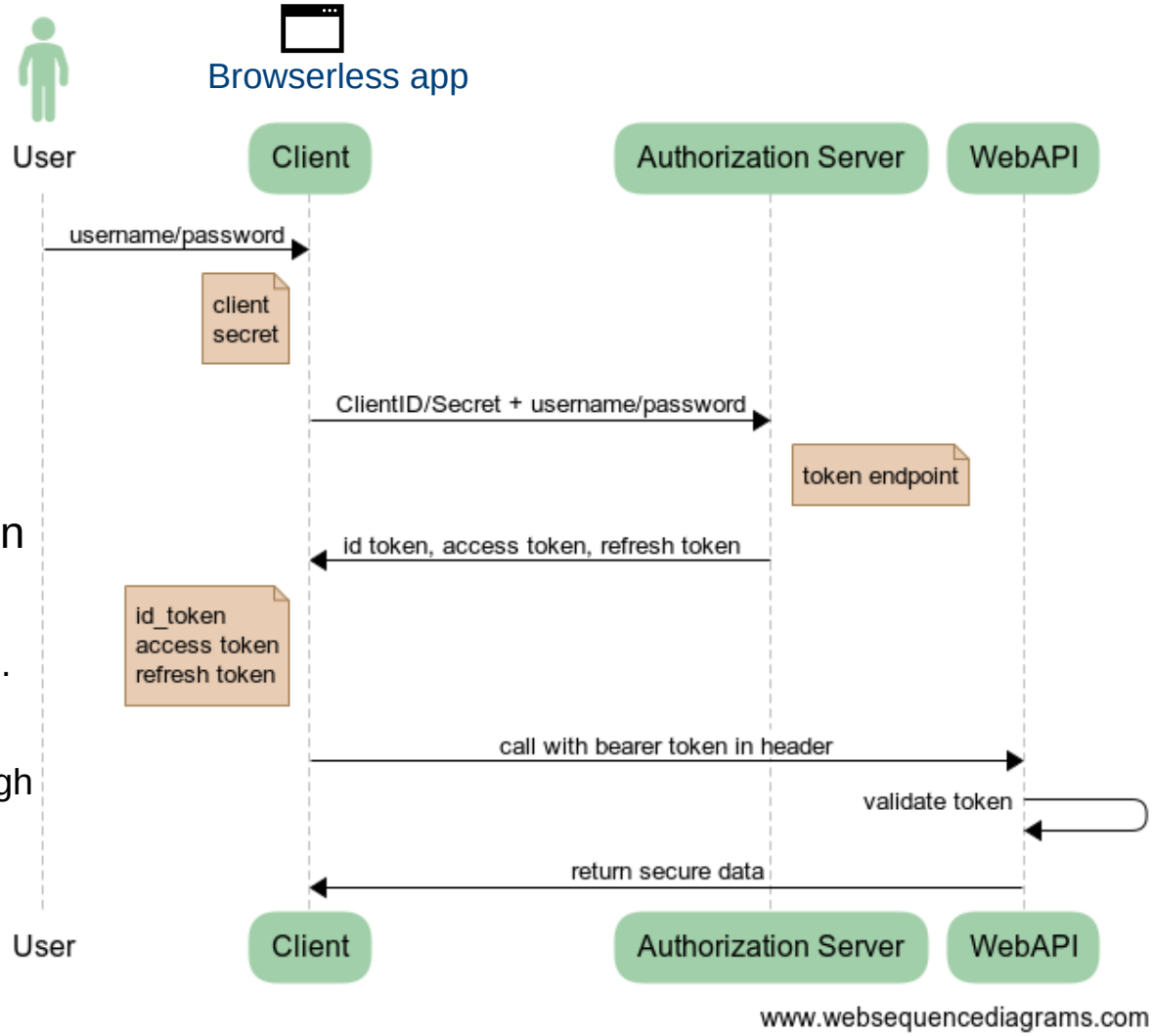
- Requires more complex server-side management and infrastructure!

- Refresh token reuse detection
  - Reuse of a previous RT
  - Invalidated already previous RT
  - Invalidate current RT
  - Invalidate access token.

- Grace period
  - For users with connection problems

Resource owner password credentials

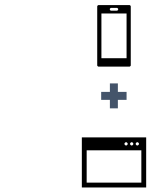
- ROPC: DO NOT USE
  - “maybe in DevOps”
- Alternatives are available and recommended
- Requires high-degree of trust in client application
  - Private, trusted client application.
    - Not public one !
  - Password goes in plain text trough the client application



Device code flow

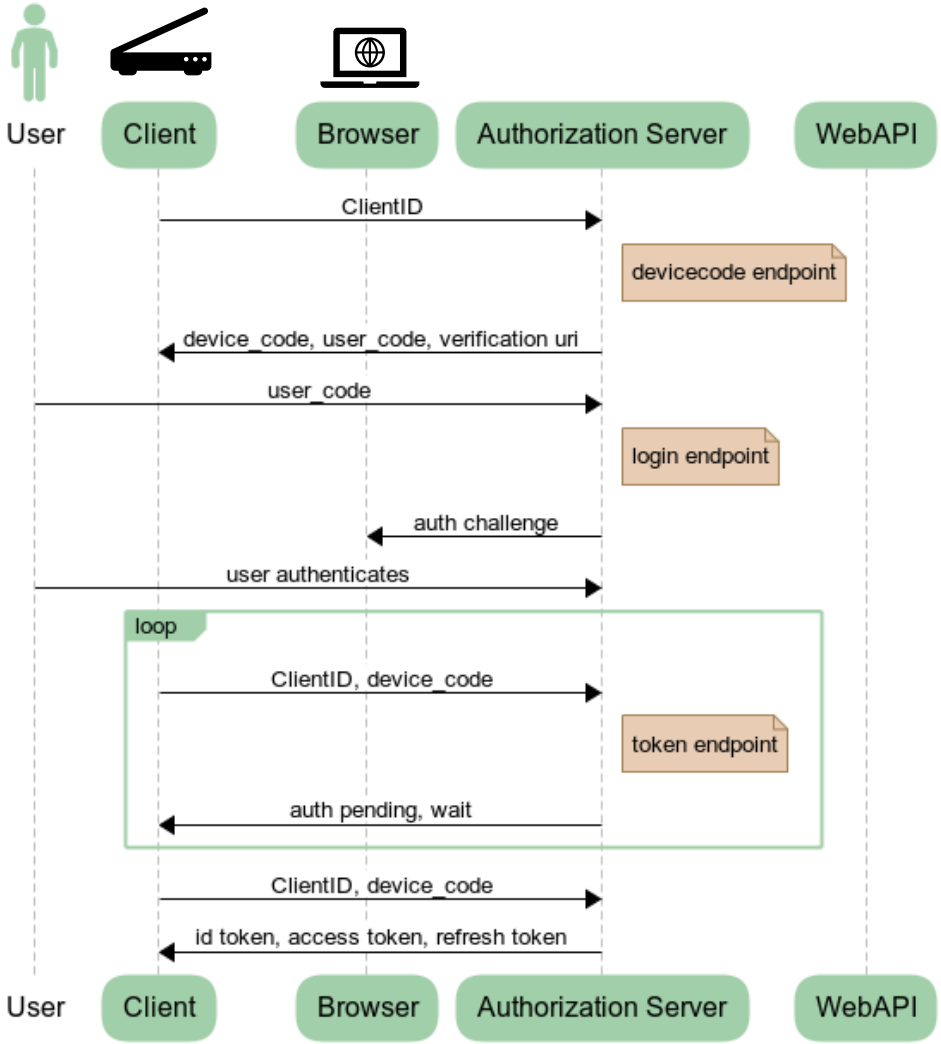
For IOT devices

- For clients
  - Browserless systems
  - Limited input device (IOT device)
- Involves 2 devices
  - User device with browser
  - Client: IOT device
- Better than ROPC



Browserless app

IOT device

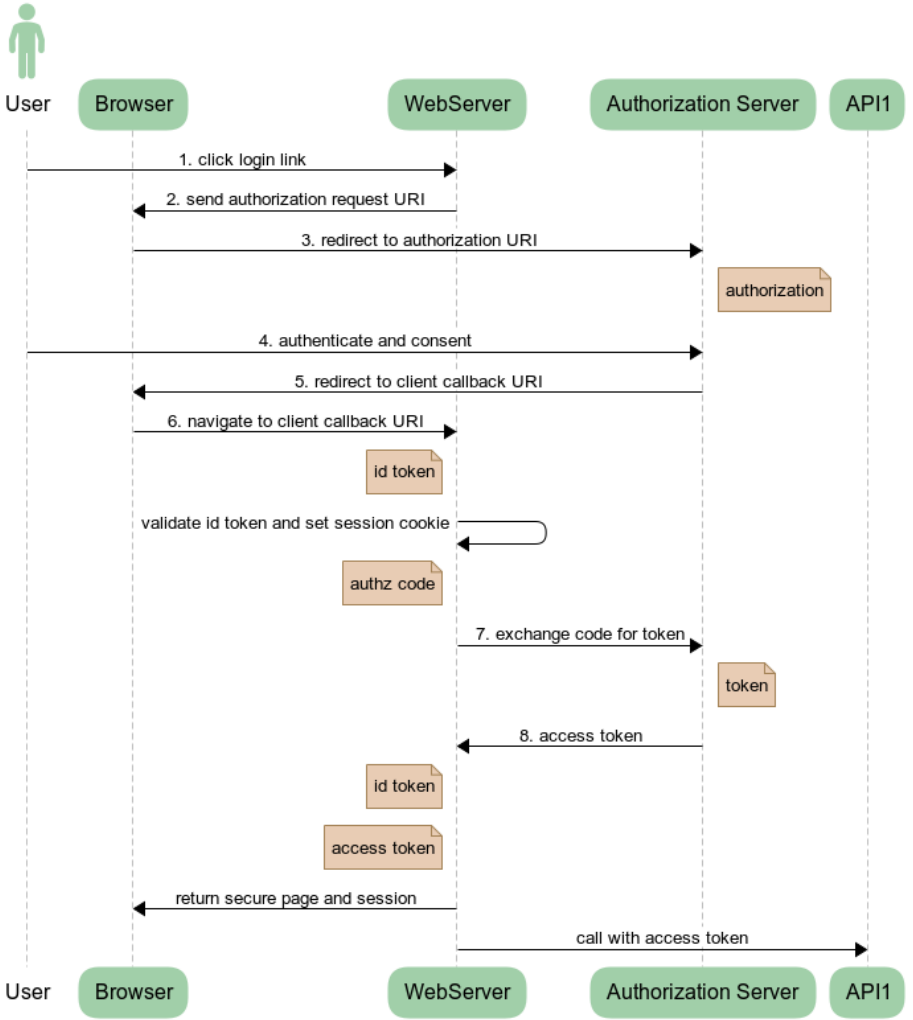


Server-side token acquisition

Server-side Web app

authz code / hybrid

- To authenticate a user
  - Protecting the web app
  - Using an externalized IdP
    - Redirect to IdP to enter credential
    - OpenID connect
  - Web app validates received id token
- To call web api on behalf of user
  - Web apps that call web APIs are confidential client applications
    - Application secret or certificate
  - Web app authorizes itself AND acts for the user
- Also used in web application firewalls and micro-service platforms
  - WAF stores the access tokens (the real meat)
  - Browser gets a cookie



www.websequencediagrams.com

Machine 2 machine, without user: Oauth 2.0 client credentials grant flow

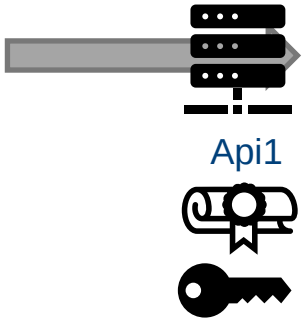
interactions in name of application accounts

consumer tier

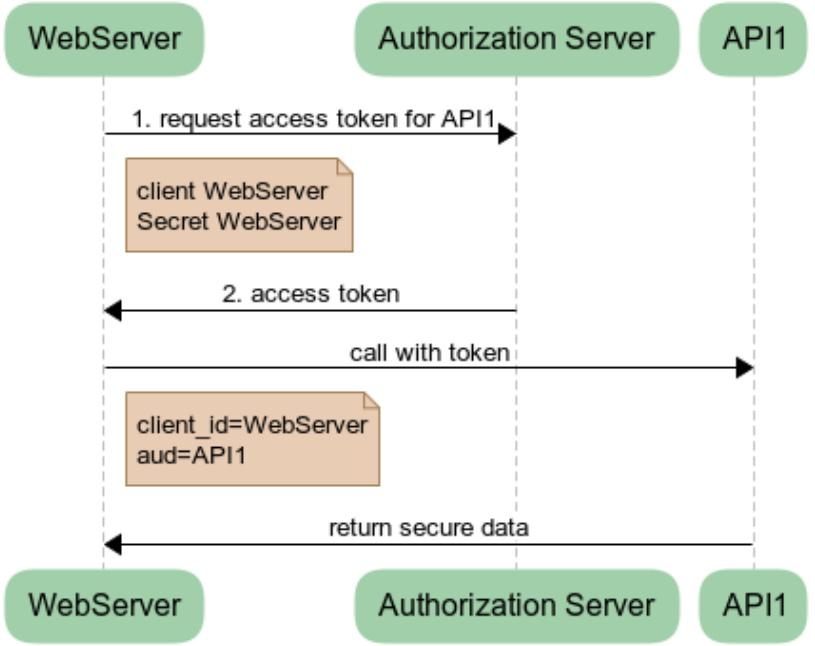
server-side daemon

tenant-side daemon

daemon web app

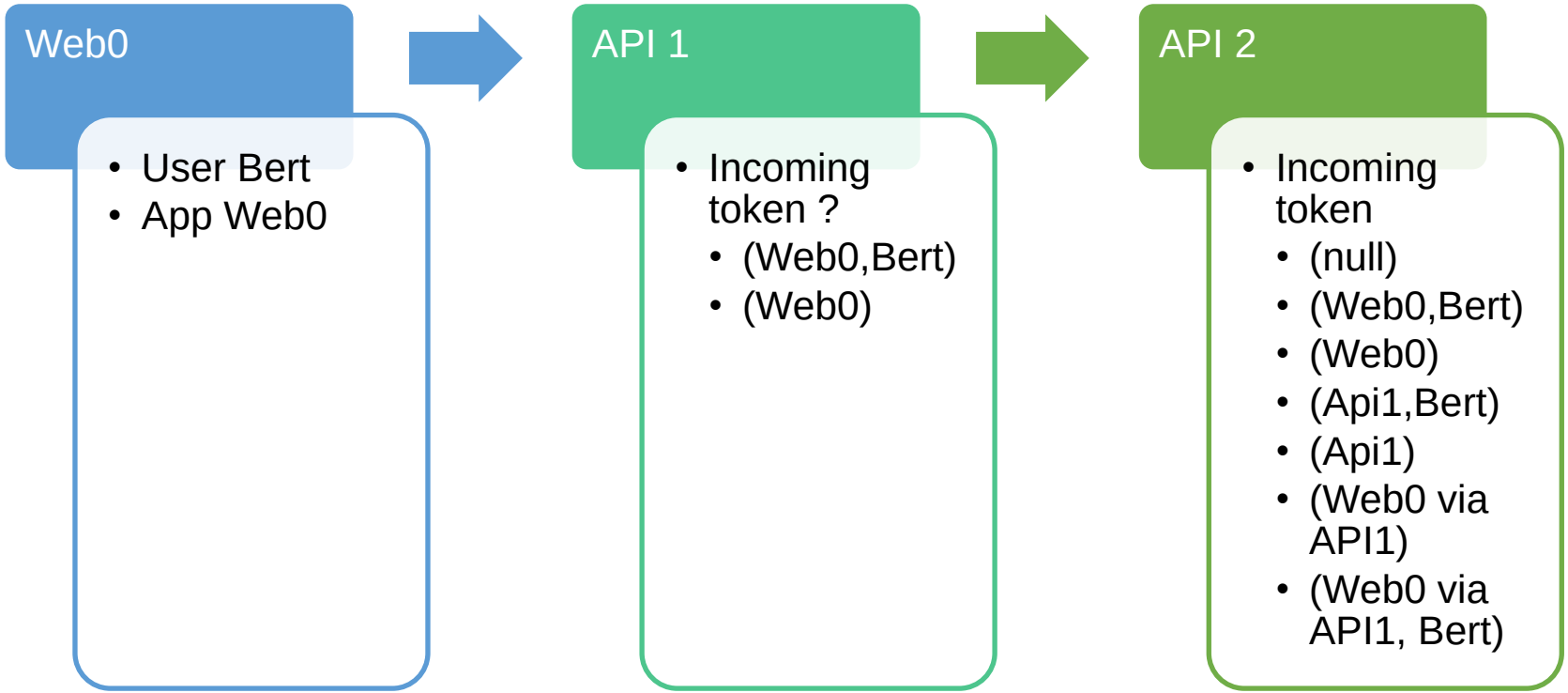


› *“two-legged Oauth”*



Calling downstream API’s
Delegation and impersonation

Calling a downstream API
Conceptual identity flow variations with proof



Calling a downstream API

WebApp → API1 → API2 → external API

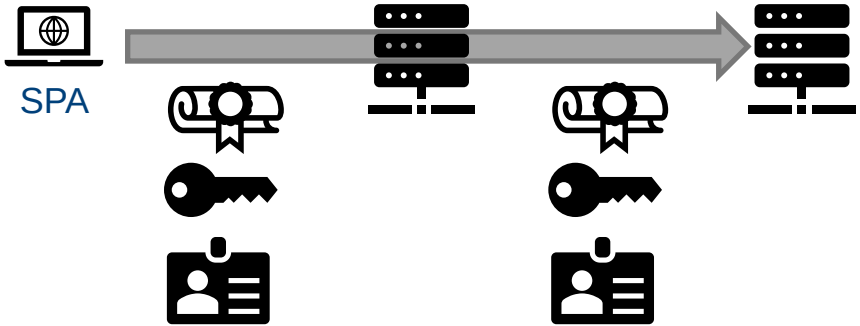
- Impersonation: When Principal A impersonates Principal B
  - A is given all the rights of B
  - A is indistinguishable from B
  - A = B
- For receivers of a token coming from A, they think they are dealing with B

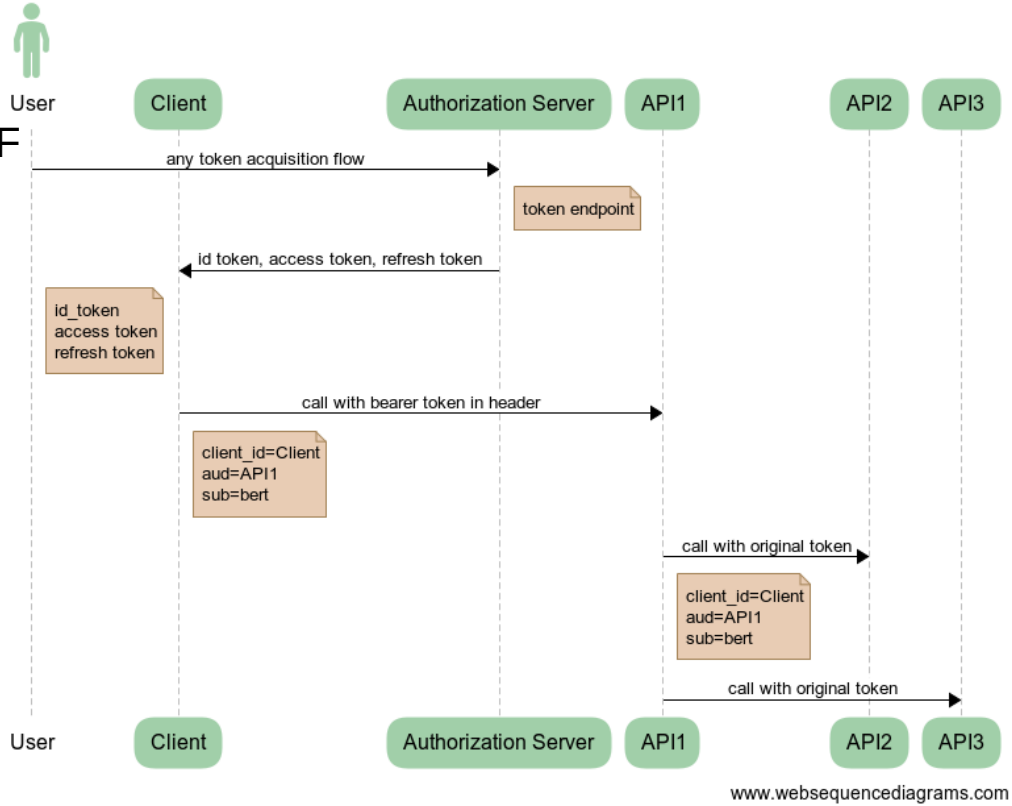
- Delegation: A still has its own identity separate from B
  - B may have delegated some rights to A
  - Actions taken by A representing B
  - A is an agent for B
- Tokens are composite token containing both the subject and actor (subject token B and actor token A)

Simple forwarding

Just forward the token

- Often used when API1 is a gateway / WAF
  - Audience per downstream API.
    - GW/WAF just forwards
  - OR all APIs are composite audience
    - All downstream APIs are API1 (depicted)
- No intermediate call to authz server



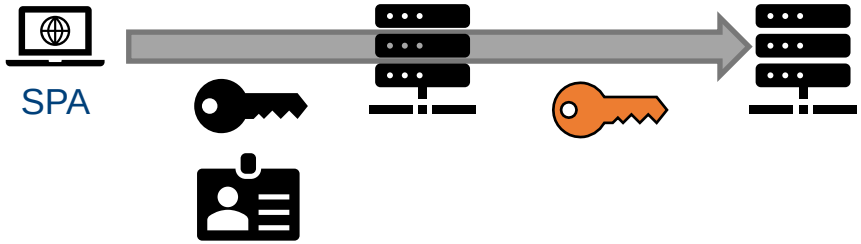


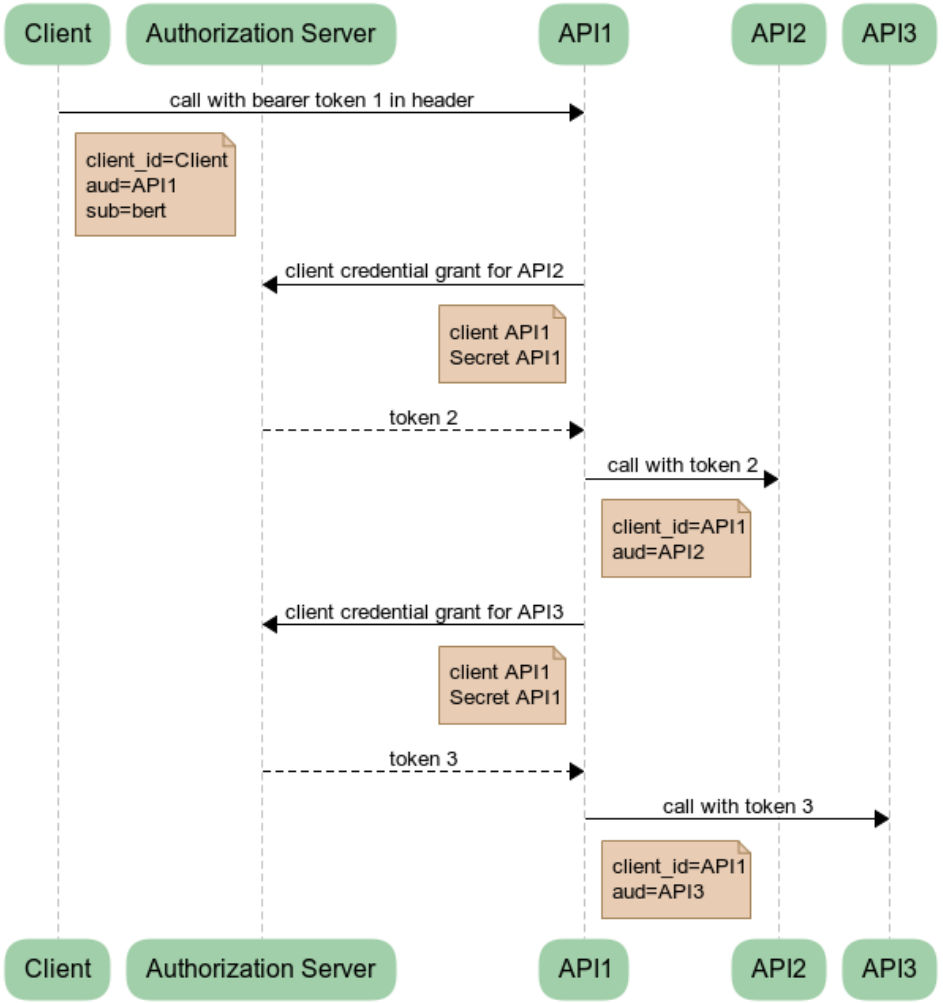
KU LEUVEN DistriNet

Calling a downstream API

Just call as app without user id

- Get new token from authz server
  - Specific for API1 as client
  - Specific for API2 as audience(target)
- Using client credentials grant flow
- Using client id and secret of the middle tier API1.
  - Acquire token 2 for API2
  - Acquire token 3 for API3
- Token can be reused within TTL
  - Then use refresh token.



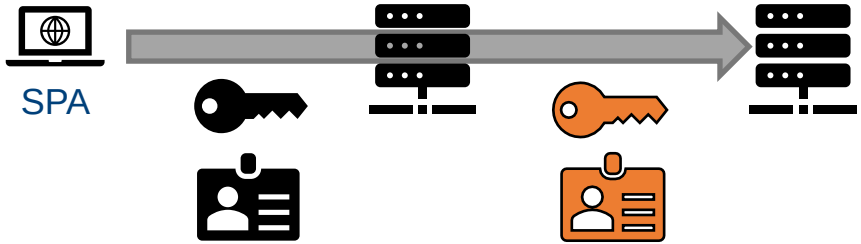


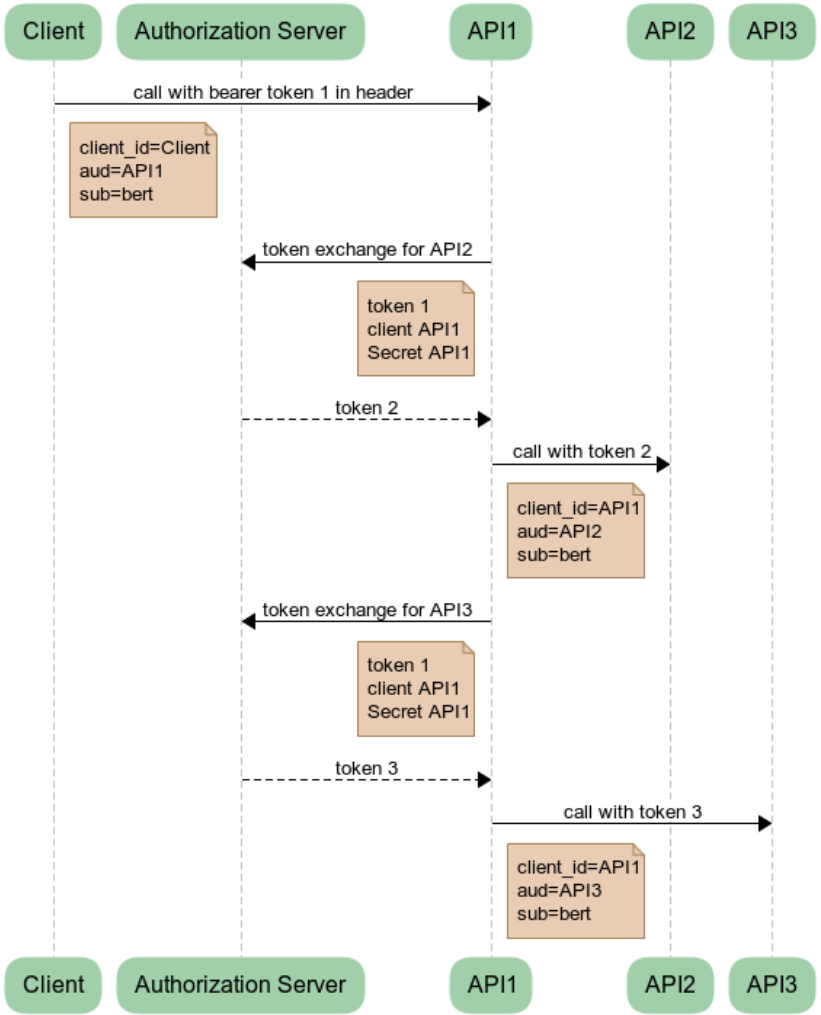
www.websequencediagrams.com

Calling a downstream API

Token exchange flow

- Get new token from authz server
  - Specific for API1 as client
  - Specific for API2 as audience(target)
  - Containing user id and user claims (id token)
- Called API 1 becomes calling client to API2
  - Uses received token to exchange for a new token containing
    - Subject (user identity)
    - New Calling Client API1
    - New Called Audience API2





Calling an external API out of your admin domain

Retrieve key or credential from a secrets store

- E.g. AWS secrets manager
  - Requires AWS credentials
- E.g. Azure Key vault
  - REST API
  - Protected by Azure AD
    - Authn and access policies
  - Additionally and optionally
    - Protected by firewall (IP)

Fighting your daemons
Client-side and server-side

Tenant-side daemons
The good, the bad and the ugly

User, app and permission management

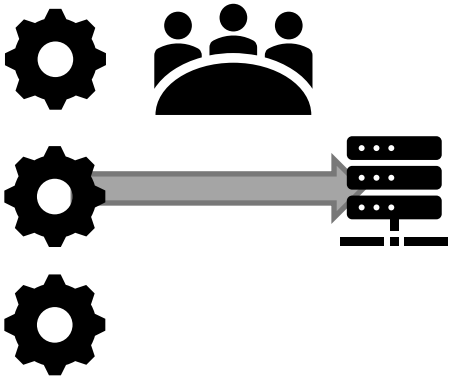
In externalized IAM systems.

- Users
  - Can be part of a group
    - E.g. group tenant A
  - Can have application roles
    - E.g. tenant-admin
  - Can have custom permissions
    - E.g. {manages: TenantA}
- (Client) Applications
  - Limited set of attributes
  - No groups
  - Permissions depend on specific IAM solutions
    - Fixed scopes per API
    - Flexible scopes per API
  - Not as easy to manage as users

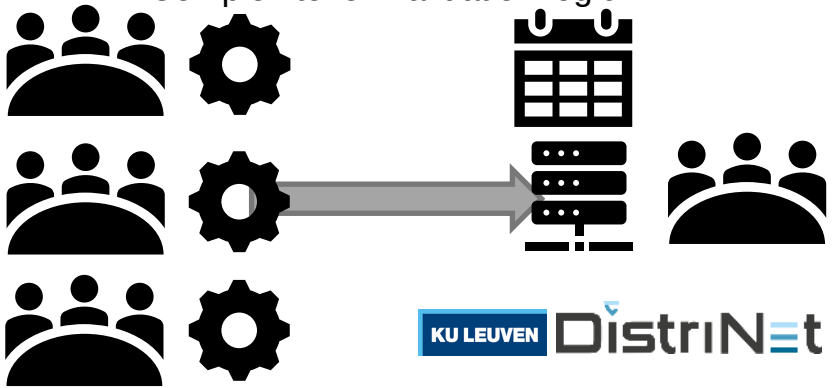
Tenant-side daemons via client credentials grant

Managed by the IAM system of the tenant or the provider

- Single-tenant API
  - IAM of provider manages all users and client apps (no redirect to tenant IAM)
  - Tenant gets client id and client secret for their client daemon from provider



- Multi-tenant API
  - IAM system of tenant is used to authenticate and authorize users and applications
  - Provider trusts tenant IAM system and tokens to do operations within the data space of the tenant.
    - Complex token validation logic !



KU LEUVEN DistriNet

Tenant-side daemon: how to manage 200 daemons ?

How to manage, authenticate and authorize tenant-side daemons ?

- With an active user
  - As a user: auth code + pkce
    - When bootstrapping the daemon
    - Via browser
      - Access token
      - long term refresh token
    - Groups ! User permissions !
    - **The UGLY**
  - As an application:
    - client credentials grant
    - with interactive admin consent before
- Without an active user
  - As an application: client credential grant
    - The (lesser) GOOD
      - Apps with only coarse grained scopes (APIs)
    - **The GOOD**
      - Apps can get fine-grained permissions
      - “data-reader”, “data-writer”, “tenant4”
  - As a synthetic user
    - ROPC ☹
    - Username and password also stored
    - User can do more than App in enterprise!
    - **THE BAD:** fully trusted client ?

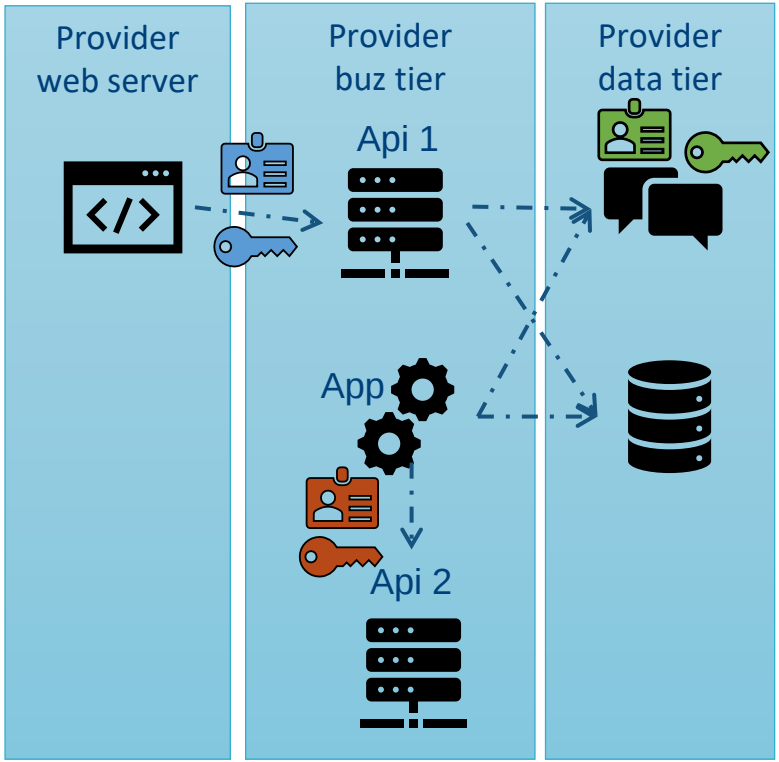
Provider-side daemons:

Don’t divide and conquer
United we stand

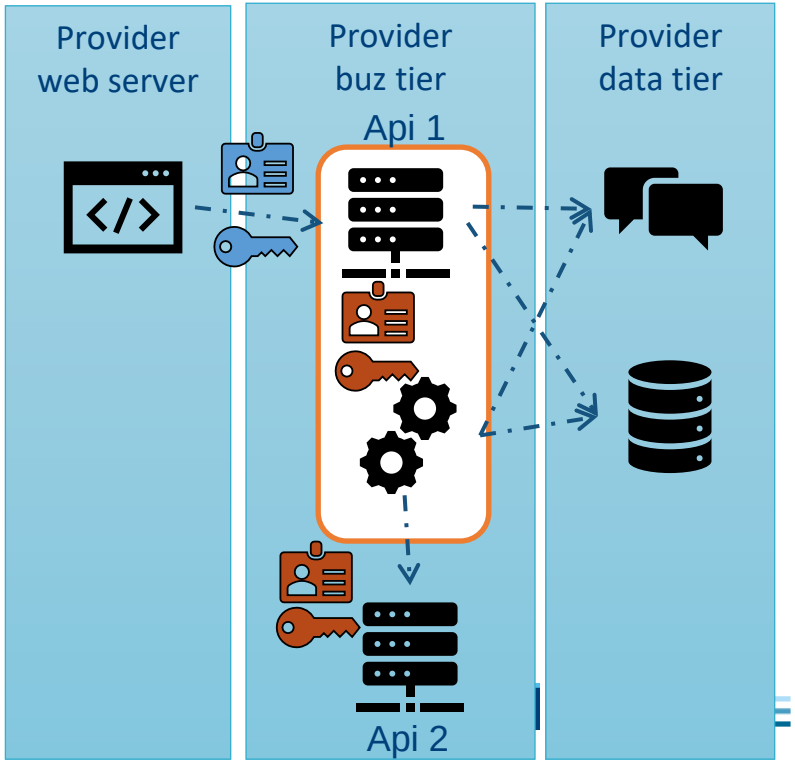
User:WebApp→API1→Queue→Daemon→API2 (id?)

Queue = communication channel with bearer token alike auth ?

› Separate applications and clients



› Composite app with shared token



Compiling variations: flows, consumers and technologies

Which flow supports which token ?

Access token and/or id token

Flow/Grant type	Access token	ID token
Implicit	V	V
Authz code	V	V
Authz code + PKCE	V	V
Hybrid	V	V
ROPC	V	V
Token exchange	V	V
Device code	V	V
Client credentials	V	X

Which flow for which client type ?

Client (Consumer)	Access token	Id token
Server-side web app	Authz code flow	
SPA	Authz code + PKCE	
Native (mobile, desktop)	Authz code + PKCE	
Fully trusted client	(ROPC)	
Daemon	Client credentials (shared) token exchange	(ROPC) (shared) token exchange
API	Client credentials, token exchange	Token exchange
IOT device	Device code	

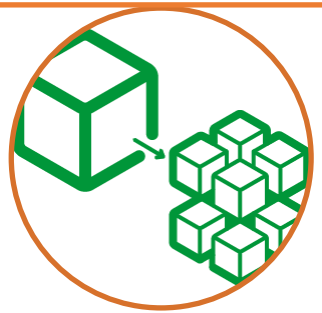
Flow/Grant overview

And their implementations in (some) technologies and managed services

Technology	KeyCloak	IdentityServer	AzureAD	Cognito	Auth0	Okta
implicit	V	V	V	V	V	V
Authz code	V	V	V	V	V	V
Authz code+PKCE	V	V	V	V	V	V
Hybrid flow	V	V	V		V	
Client Credentials	V	V	V	V	V	V
Token Exchange	V (loosely)	V(delegation)	V (OBO)			
ROPC	V	V	V		V	V
Device code		V	V		V	
RT rot. (single use)	+/- (revoke)	V	X	X	V	V

Ongoing DistriNet research & Open Challenges

Dynamo: Monitoring and analytics of application-level authentication and authorization in cloud apps



Reference architectures

- Variations of cloud application architectures
- Microservice based
  - web apps
  - api's
    - Sub-services/components
  - Background worker
- Client variation

Monitoring probes

- Trace incoming http traffic
- Trace outgoing http traffic

- Implementation-level
- **Non-intrusive pure config-based**

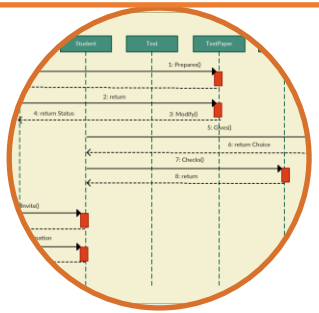
- **Scalable and performant monitoring**

Communication channel

- **Push-based collection of monitoring data in back-end**

- **Synchronous and asynchronous**
- **Single and Batch**

- **Extensive performance analysis**



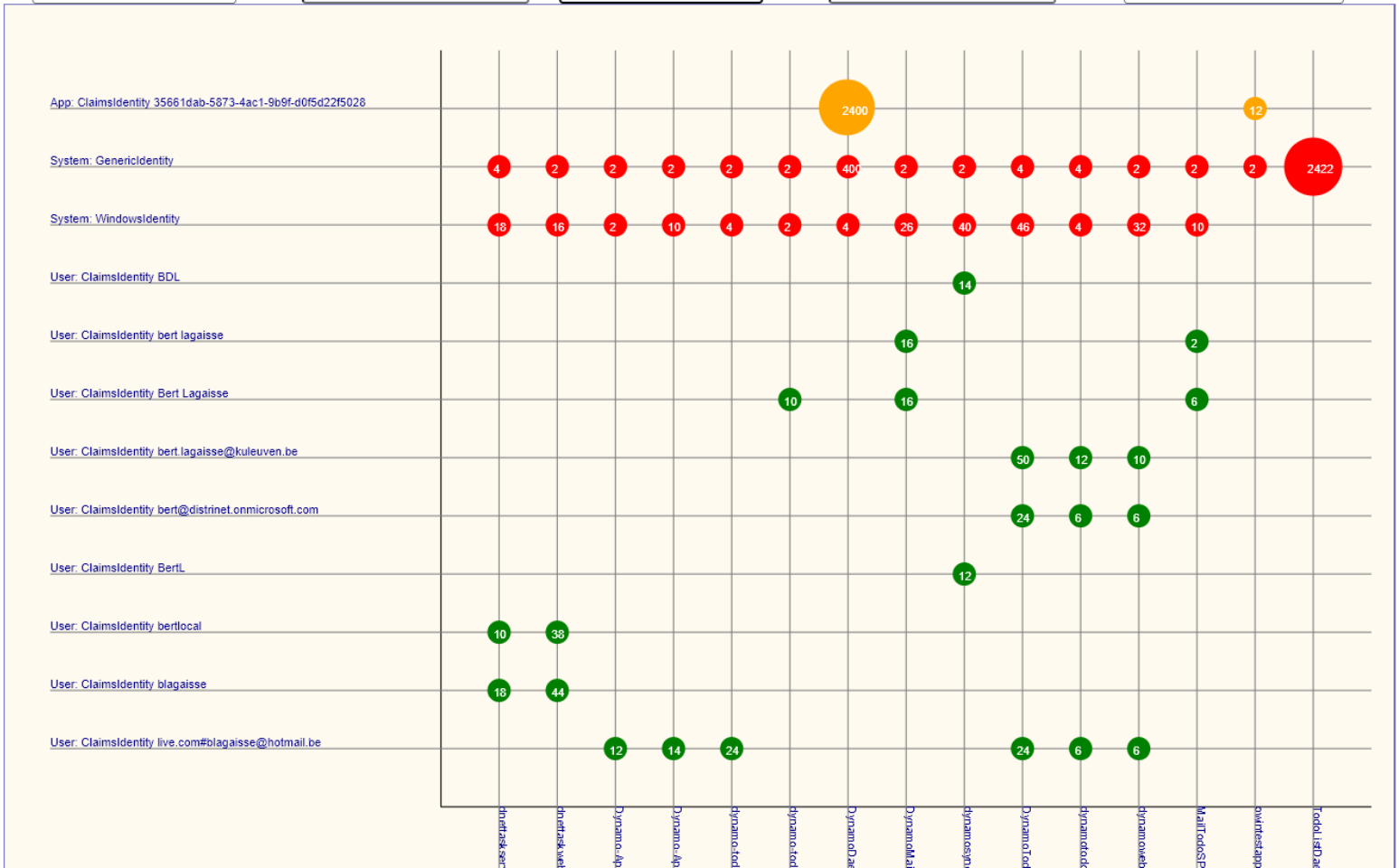
Off-line *Analytics* and **visualization**

- http-interactions in 1 service
  - All incoming & all outgoing
- Correlations within a trace
- **Distributed flows of Micro-services**
- **Sub-components (controllers)**
- **External services (azure)**
- **Authn & Authz statistics**
- **Timing**

Blazor Tracer

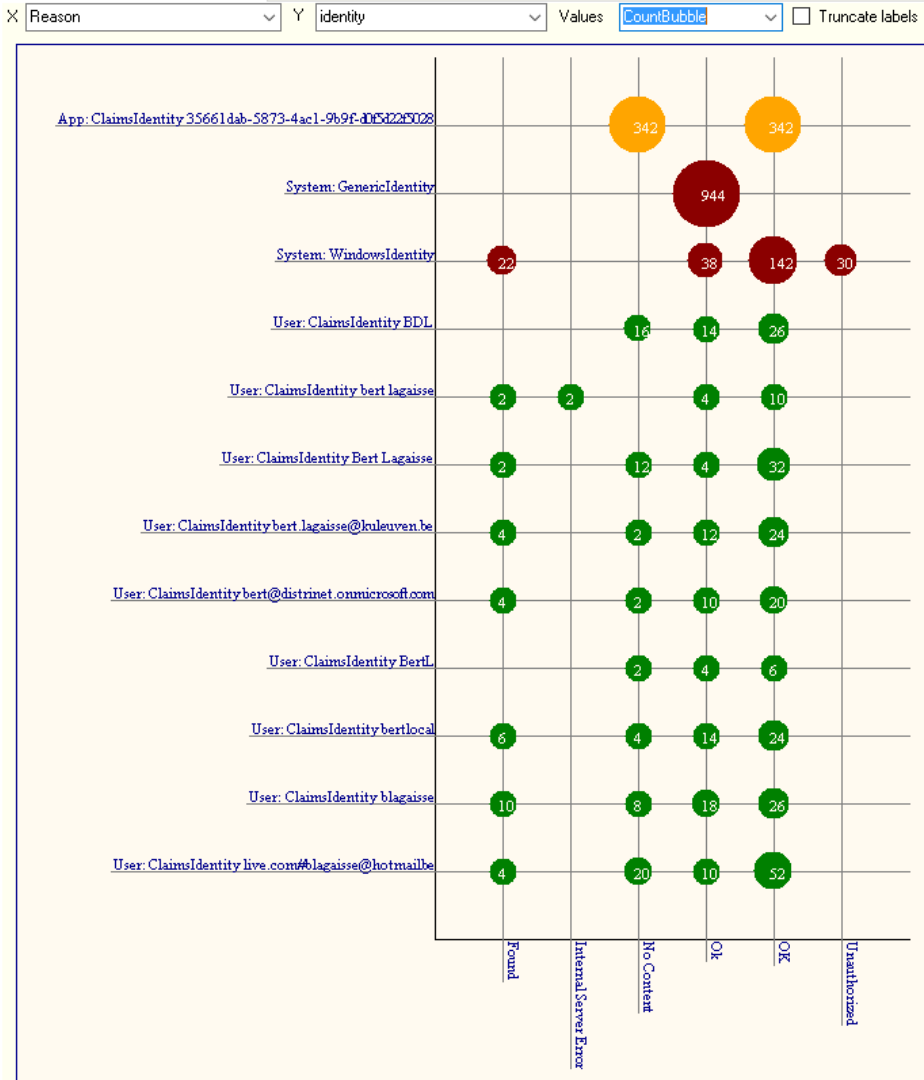
From 01-02-2021 To 23-02-2021 Scope App Fetch Data Traces 2/1/2021 9:00:13 AM (2) Analyse

X action XFilter Y identity YFilter Values CountBubble Scale Logarithmic View



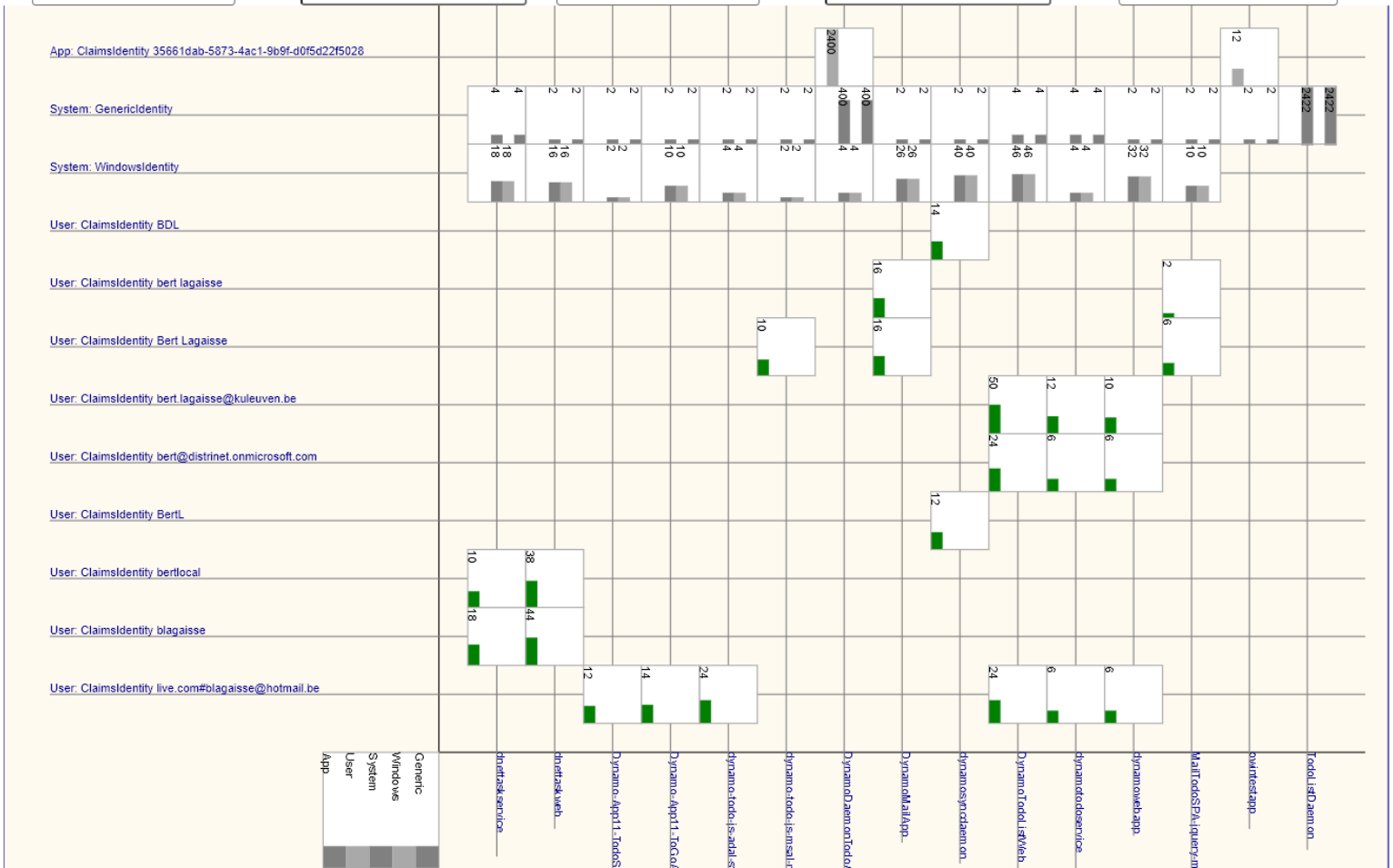
Demo and examples Monitoring for AAA

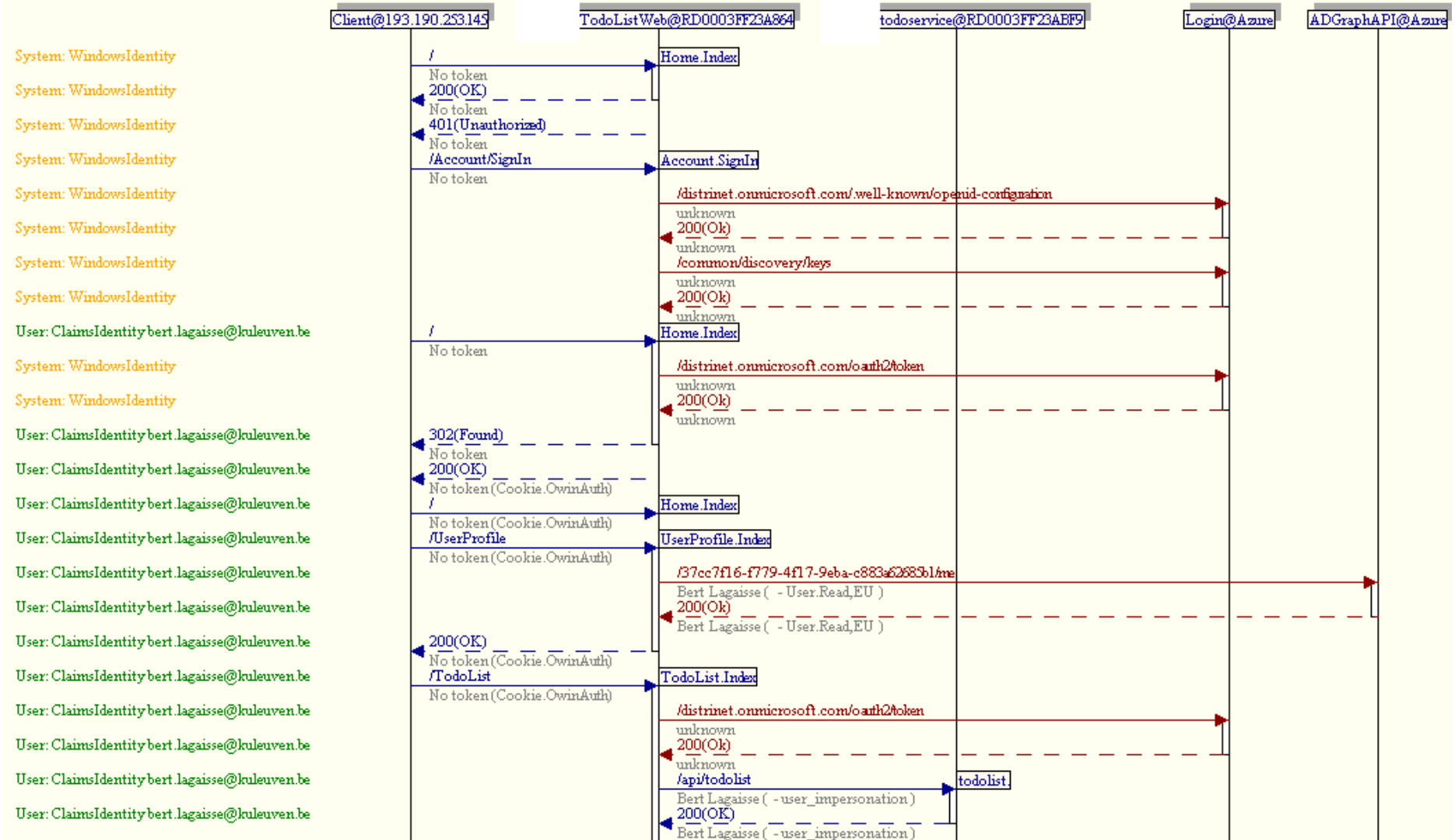
- *Which identities did actions that resulted in “unauthorized” and in which quantity? (1)*
- Which identity is used in which micro-service for which action.
  - Problem to find: are certain actions executing under the wrong kind of principal (System, app or user)
- Which tokens contain which identities (app and user) and result in successful authorization
  - Which don’t ?
- Which identity is used by micro-services when calling other micro-services
- *How are an identity and its permissions propagated throughout the architecture towards downstream microservices ? (2)*



Blazor Tracer | From 01-02-2021 | To 23-02-2021 | Scope App | Fetch Data | Traces 2/1/2021 9:00:13 AM (2) | Analyse

X module | XFilter | Y identity | YFilter | Values IdentityDivision | Scale | Logarithmic | View




Client@193.190.253.145
TodoListWeb@RD0003FF23A864
todoservice@RD0003FF23ABF9
Login@Azure
ADGraphAPI@Azure
System: WindowsIdentity
/
Home.Index
No token
200(OK)
No token
401(Unauthorized)
No token
/Account/SignIn
Account.SignIn
No token
/distrinet.onmicrosoft.com/.well-known/openid-configuration
unknown
200(Ok)
unknown
/common/discovery/keys
unknown
200(Ok)
unknown
User: ClaimsIdentity bert.lagaisse@kuleuven.be
/
Home.Index
No token
/distrinet.onmicrosoft.com/oauth2/token
unknown
200(Ok)
unknown
302(Found)
No token
200(OK)
No token (Cookie.OwinAuth)
/
Home.Index
No token (Cookie.OwinAuth)
/UserProfile
UserProfile.Index
No token (Cookie.OwinAuth)
/37cc7f16-f779-4f17-9eba-c883a62685b1/me
Bert Lagaisse (- User.Read,EU)
200(Ok)
Bert Lagaisse (- User.Read,EU)
200(OK)
No token (Cookie.OwinAuth)
/TodoList
TodoList.Index
No token (Cookie.OwinAuth)
/distrinet.onmicrosoft.com/oauth2/token
unknown
200(Ok)
unknown
/api/todolist
todolist
Bert Lagaisse (- user_impersonation)
200(OK)
Bert Lagaisse (- user_impersonation)

Interesting first conclusions in the project: Shift in original focus of application providers…

From: user-behavior analysis

- Did the user misbehave ?
- Are attackers trying to break-in into my web application ?
- Find security breaches

To: application-dev analysis

- Did the application dev use the right authentication and authorization in the right places ?
- Detect weak spots in the reversed architecture
  - “Because we don’t know our actual, explicit security architecture”
  - “Because we didn’t create a security architecture up-front, or after the implementation”

When token is presented to API
Open challenge: How did you get that token ?

- Token does not specify how it was obtained:
  - Implicit flow
  - Authz code
  - Authz code + pkce
- Interesting for API providers
  - Monitoring, management, certification
  - How many legacy clients still active ?
  - How active are these
  - Is that dev still using that implicit flow !?
- Authorization server knows
  - Self-managed: check logs
  - As-a-service … ?

- Answers from IAM-aas providers (Twitter):
  - “Not standardized afaik. Your IdP may add it as a claim if you control it to that degree. “
  - “You can’t derive that information from the token alone I’m afraid. Is there a reason you need to know this?”
  - “Interesting one. Pretty sure our implementation (PingFederate) would provide a technical means to accomplish this by mapping the response type into a token claim, but it's not a use case I've heard of before. May I ask why you'd want to do this?”
  - “In Azure AD, you might able to see that client credentials was used for an access token as the aud would be something ending with /.default. No way to find it out for others AFAIK.”

After the holidays:
Advanced application-level access control research
leveraging ABAC and PBAC

Thank you!

https://distrinet.cs.kuleuven.be/