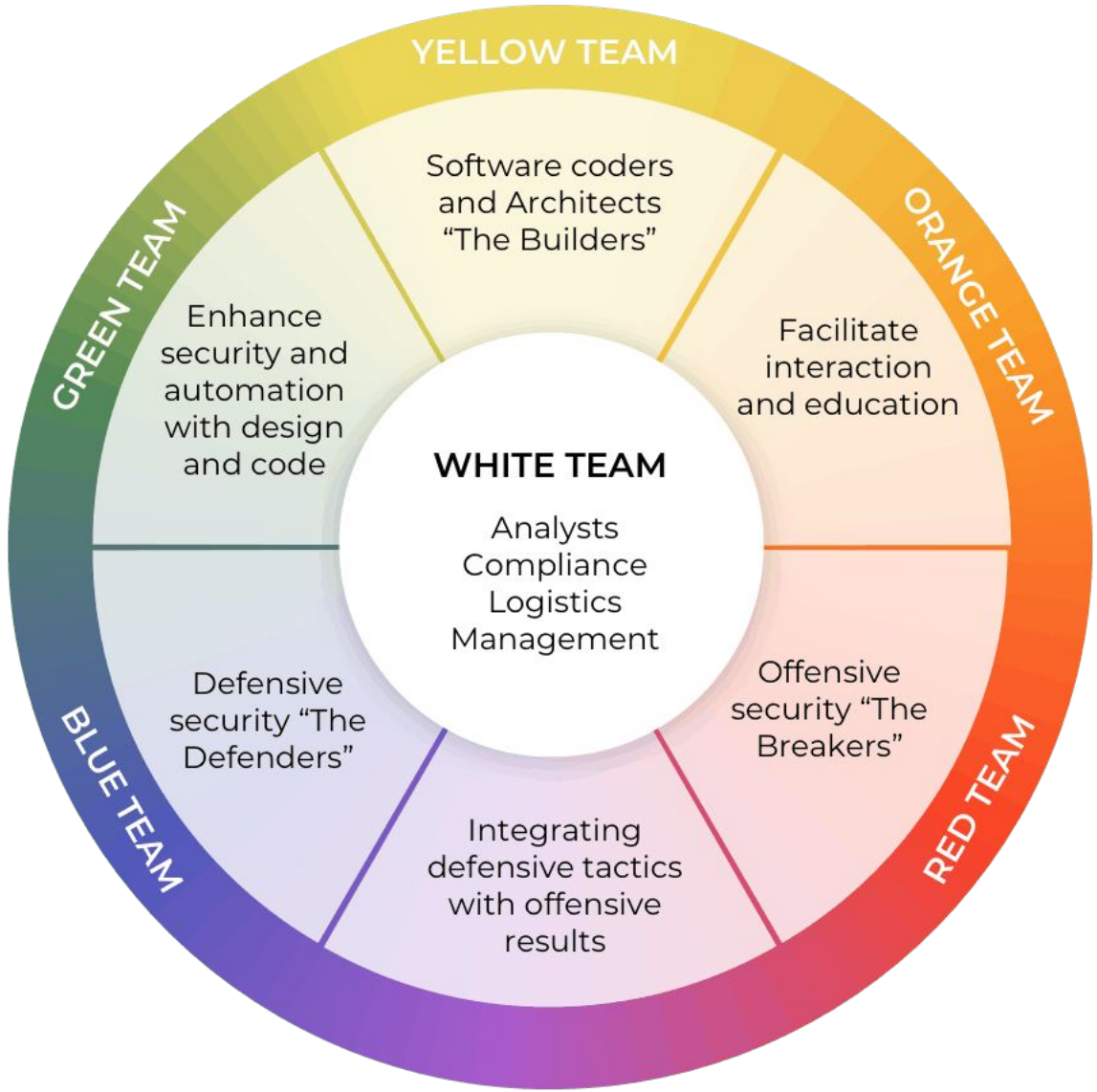
YELLOW TEAMING - III


YELLOW TEAM
Software coders and Architects "The Builders"
ORANGE TEAM
Facilitate interaction and education
RED TEAM
Offensive security "The Breakers"
Integrating defensive tactics with offensive results
BLUE TEAM
Defensive security "The Defenders"
GREEN TEAM
Enhance security and automation with design and code
WHITE TEAM
Analysts
Compliance
Logistics
Management

YELLOW TEAM

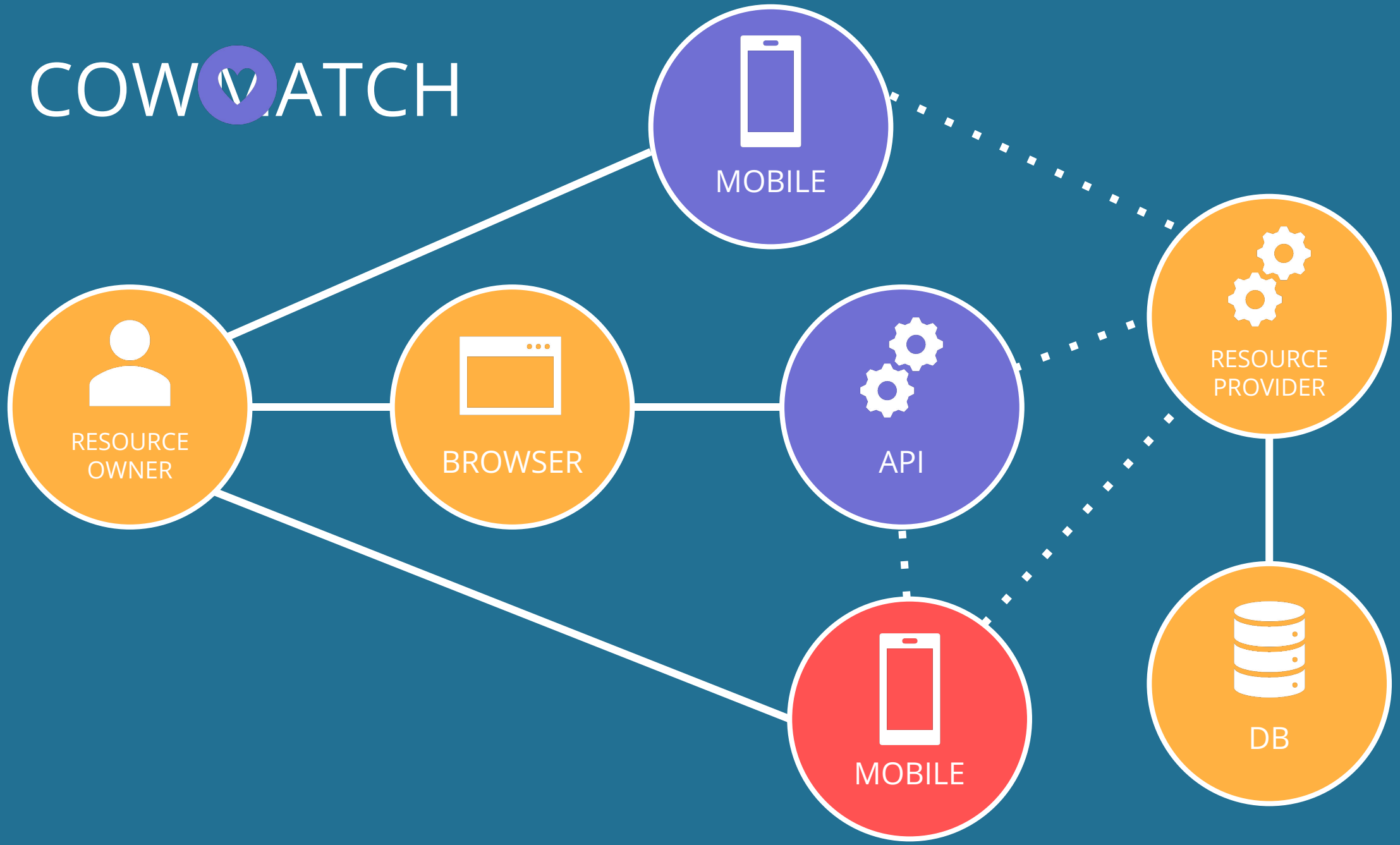
- Software Builders
- Application Developers
- Software Engineers
- System Architects

INPUT SANITIZATION

DON’T TRUST THE CLIENT

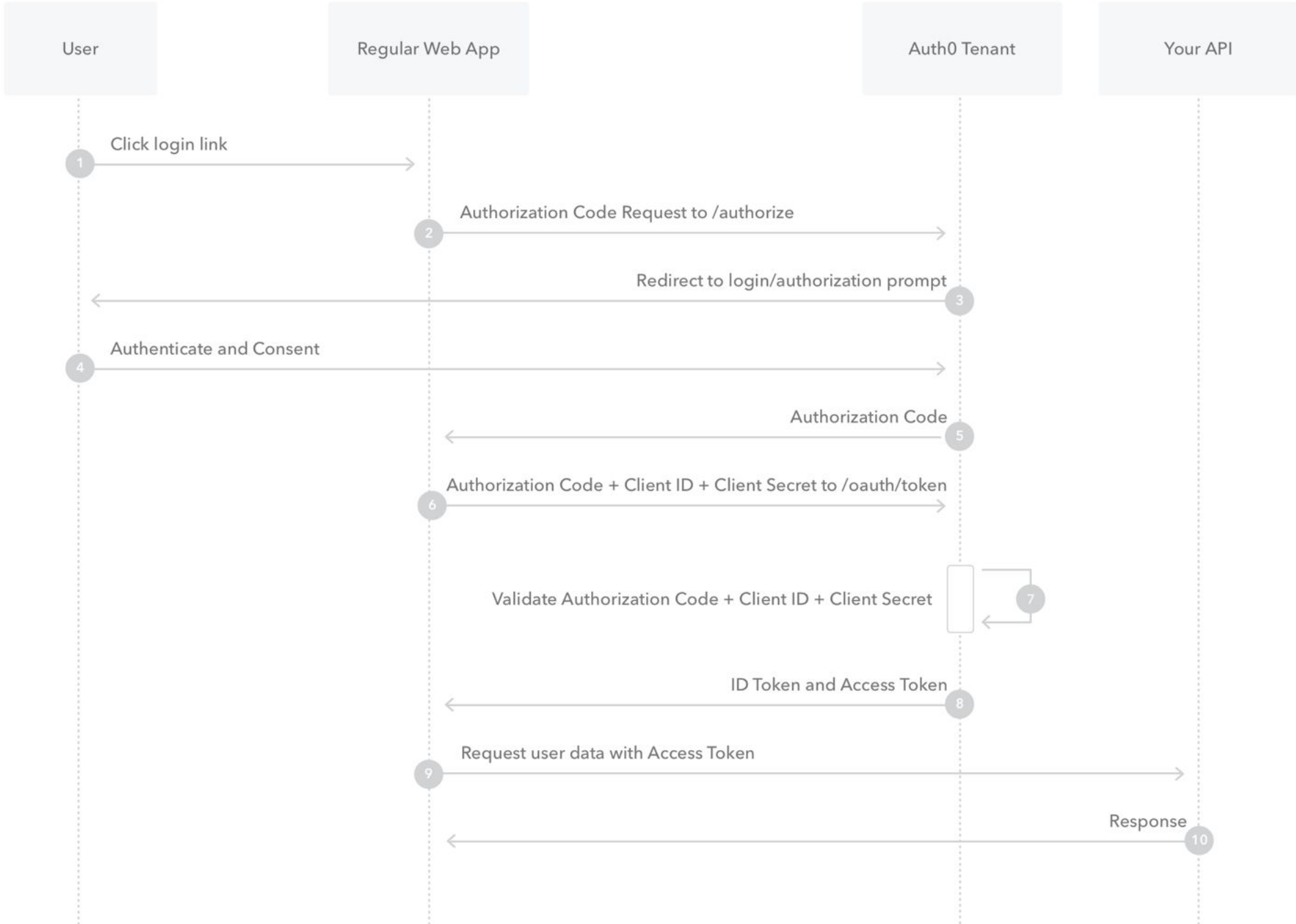
UPDATE YOUR SHIT

Wijsheid van Beckers™


COWATCH
MOBILE
RESOURCE OWNER
BROWSER
API
RESOURCE PROVIDER
MOBILE
DB

MACHINES COMMUNICATING ON BEHALF OF HUMANS

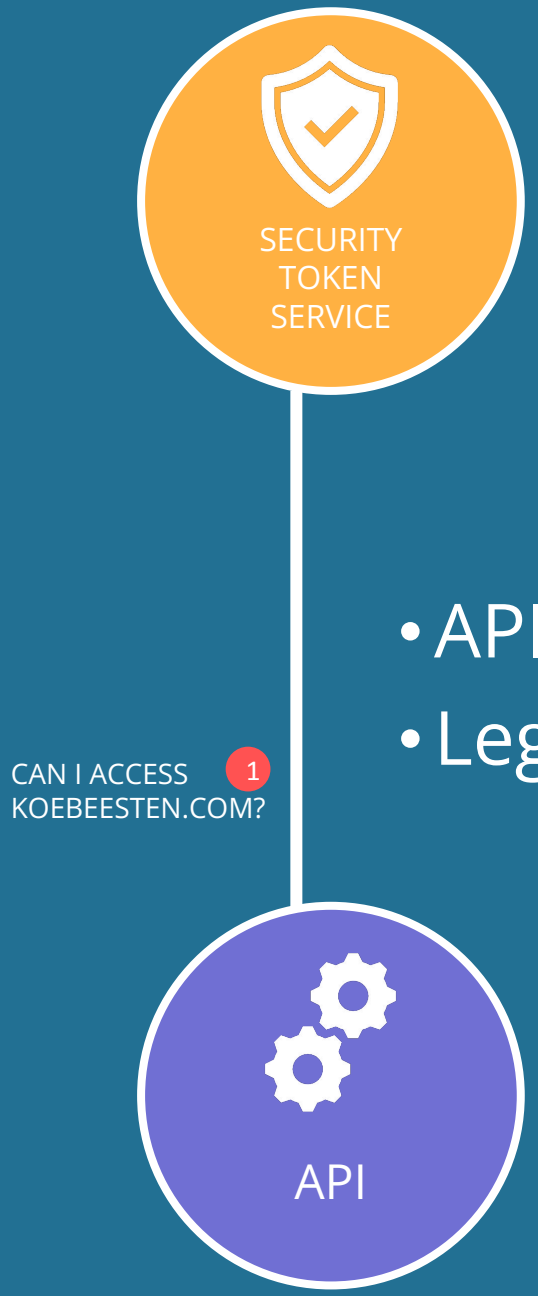
SSO!


User
Regular Web App
Auth0 Tenant
Your API
Click login link
1
Authorization Code Request to /authorize
2
Redirect to login/authorization prompt
3
Authenticate and Consent
4
Authorization Code
5
Authorization Code + Client ID + Client Secret to /oauth/token
6
Validate Authorization Code + Client ID + Client Secret
7
ID Token and Access Token
8
Request user data with Access Token
9
Response
10

- Roll your own (please don’t)
- Use existing services (Auth0)
- Use middleware
- Use a public STS

- Keep it secret! Keep it safe!
- Bearer Token! Embrace HTTPS!
- Do not add sensitive data
- Give tokens an expiration
- Adding a secondary token verification system might be necessary
- Store and reuse, avoid round-trips

OAUTH 2.0

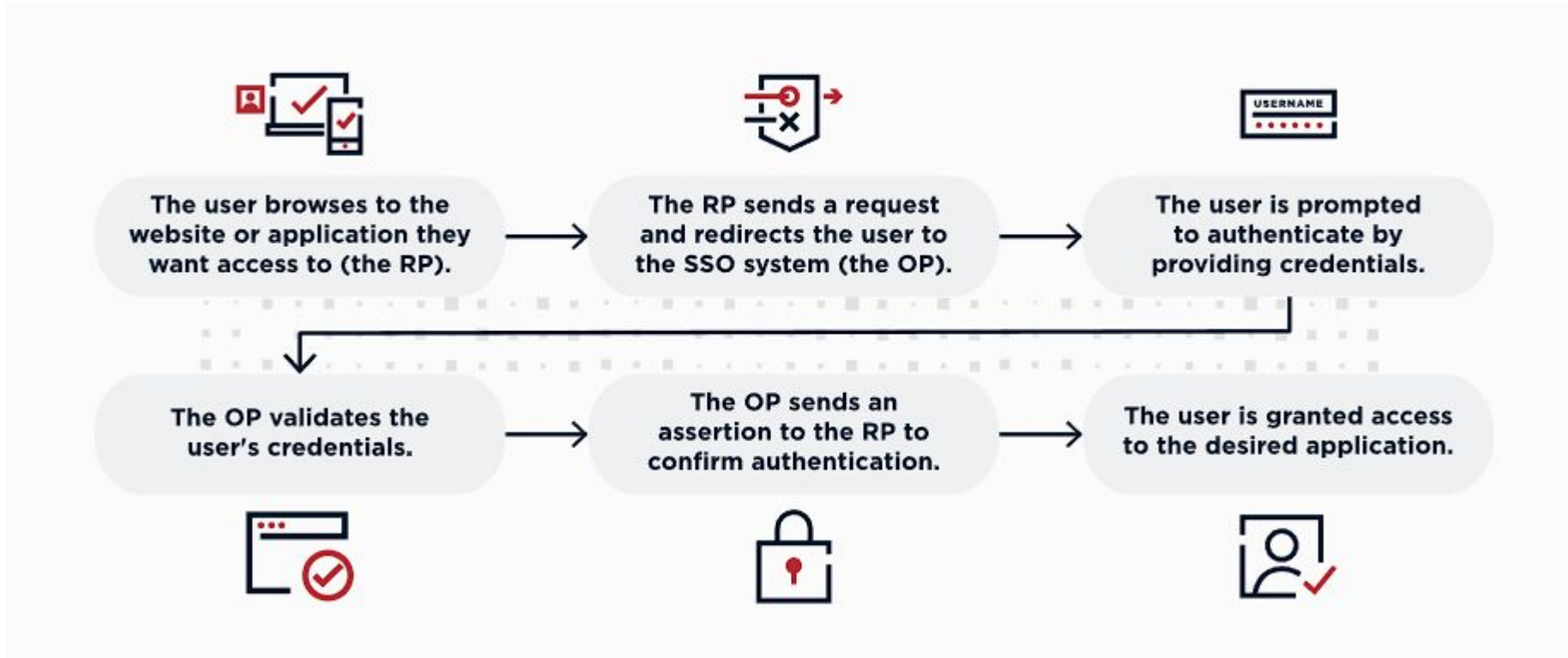


- API needs to be registered
- Legitimate redirect URI

OAUTH 2.0 FOR AUTHENTICATION

OpenID Connect

- Extension to OAuth
  - Prove who I am by proving I have access to a resource
  - ...not a house or a car!
  - ...but a token-resource that was made specifically for OpenID
- Access token -> ID Token
- “Log in with Google/Twitter/etc”
- SSO!

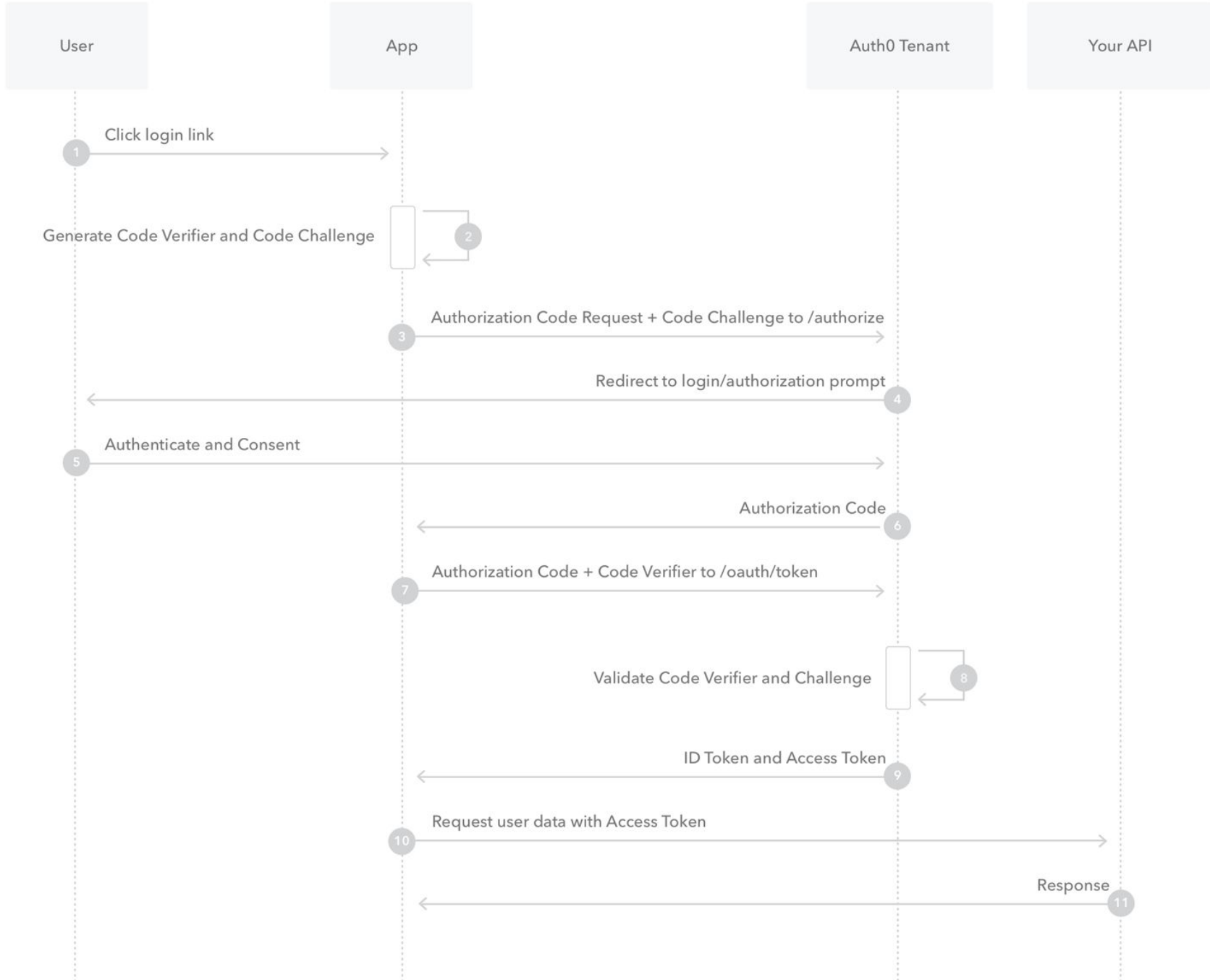

USERNAME
The user browses to the website or application they want access to (the RP).
The RP sends a request and redirects the user to the SSO system (the OP).
The user is prompted to authenticate by providing credentials.
The OP validates the user's credentials.
The OP sends an assertion to the RP to confirm authentication.
The user is granted access to the desired application.

- Cannot be trusted
- Client secret
- Redirect URI

Proof Key for Code Exchange

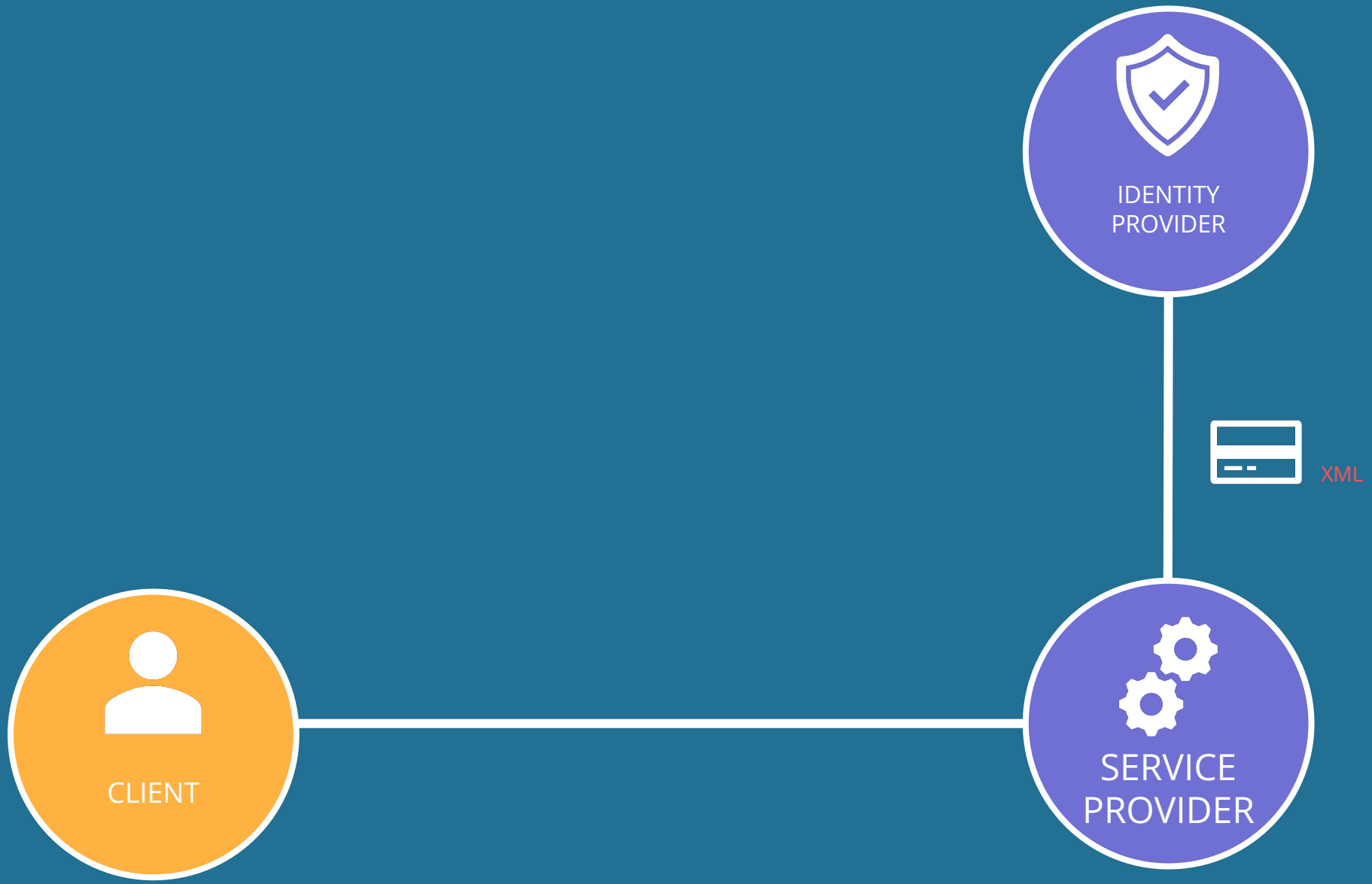
Code Challenge = SHA256(Code verifier)

- Generate code verifier and code challenge
- Code challenge used in front channel
- Code verifier used instead of client credentials
- STS gets code challenge from client, code verifier from api
- Maybe we should do this all the time?


User
App
Auth0 Tenant
Your API
Click login link
1
Generate Code Verifier and Code Challenge
2
Authorization Code Request + Code Challenge to /authorize
3
Redirect to login/authorization prompt
4
Authenticate and Consent
5
Authorization Code
6
Authorization Code + Code Verifier to /oauth/token
7
Validate Code Verifier and Challenge
8
ID Token and Access Token
9
Request user data with Access Token
10
Response
11


SPA

SAML!


IDENTITY
PROVIDER
XML
CLIENT
SERVICE
PROVIDER

SAML Example Steps

1 User logs into SSO

2 User tried to access a webpage

3 Service provider checks the users credentials with the identity provider

4 Identity provider sends authorization and authentication messages back to service provider

5 User can now log into the CRM

VARONIS

SAML!

INPUT SANITIZATION

DON’T TRUST THE CLIENT

UPDATE YOUR SHIT

Wijsheid van Beckers™

- https://pragmaticwebsecurity.com/courses/introduction-oauth-oidc.html
- Getting Started with ASP.NET Core and Oauth https://app.pluralsight.com/library/courses/asp-dot-net-core-oauth
- Securing ASP.NET Core 3 With OAuth2 and OpenID Connect https://app.pluralsight.com/library/courses/securing-aspnet-core-3-oauth2-openid-connect