Full Stack Application Development

BITS Pilani

innovate achieve lead

BITS Pilani

Pilani Campus

Module 5: Securing Applications

Agenda

- Basic Authentication
- API Authorization
- JSON Web Tokens
- OAuth
- OpenID
- HTTPS
- Common Vulnerabilities
- Cross Origin Resource Sharing

HTTP request

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.1 zlib/1.2.11
Host: www.example.com
Accept-Language: en
```

```
HTTP/1.1 200 OK
Date: Wed, 30 Jan 2019 12:14:39 GMT
Server: Apache
Last-Modified: Mon, 28 Jan 2019 11:17:01 GMT
Accept-Ranges: bytes
Content-Length: 12
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
```

If a website uses HTTP instead of HTTPS, all requests and responses can be read by anyone who is monitoring the session. Essentially, a malicious actor can just read the text in the request or the response and know exactly what information someone is asking for, sending, or receiving.

What is HTTPs ?

The S in HTTPS stands for "secure." HTTPS uses TLS (or SSL) to encrypt HTTP requests and responses, so in the example above, instead of the text, an attacker would see a bunch of seemingly random characters.

Instead of:

GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.l zlib/1.2.11
Host: www.example.com
Accept-Language: en

The attacker sees something like:

t8Fw6T8UV81pQfyhDkhebbz7+oiwldr1j2gHBB3L3RFTRsQCpaSnSBZ78Vme+DpD
VJPvZdZUZHpzbbcqmSW1+3xXGs

Question

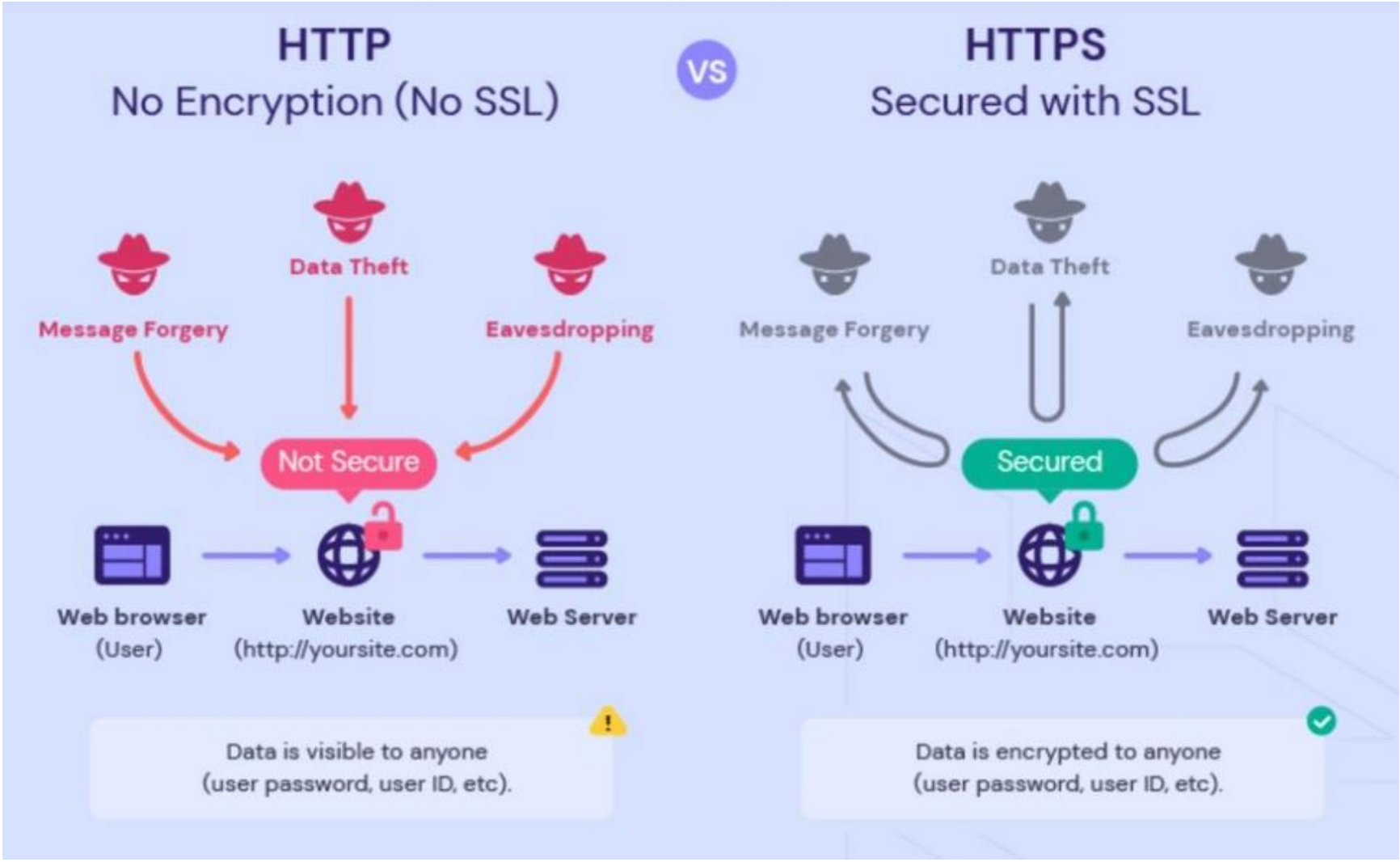
In HTTPS, how does TLS/SSL encrypt HTTP requests and responses?

Answer

TLS uses a technology called public key cryptography: there are two keys, a public key and a private key, and the public key is shared with client devices via the server's SSL certificate. When a client opens a connection with a server, the two devices use the public and private key to agree on new keys, called session keys, to encrypt further communications between them.

All HTTP requests and responses are then encrypted with these session keys, so that anyone who intercepts communications can only see a random string of characters, not the plaintext.

HTTP vs HTTPs



HTTP vs HTTP(s)

innovate achieve lead

	HTTP	HTTPS
Stands for	Hypertext Transfer Protocol	Hypertext Transfer Protocol Secure
Underlying Protocols	HTTP/1 and HTTP/2 use TCP/IP. HTTP/3 uses QUIC protocol.	Uses HTTP/2 with SSL/TLS to further encrypt the HTTP requests and responses
Port	Default Port 80	Default Port 443
Used for	Older text-based websites	All modern websites
Security	No additional security features	Uses SSL certificates for public-key encryption
Benefits	Made communication over the internet possible	Improves website authority, trust, and search engine rankings

Security Aspects

Authentication

Confirms users are who they say they are

Authorization

Gives users permission to access a resource

Question

innovate achieve lead

What is CORS (cross-origin resource sharing)?

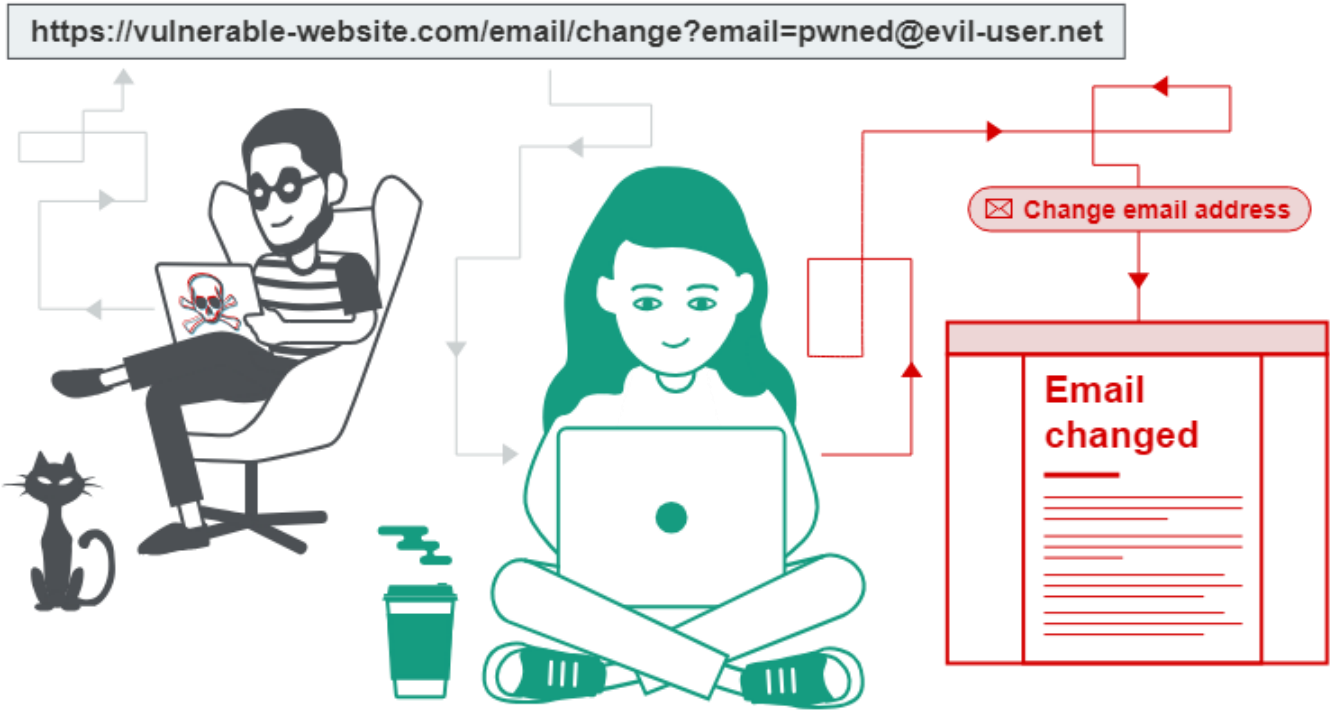
Answer

Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy (SOP). However, it also provides potential for cross-domain attacks, if a website's CORS policy is poorly configured and implemented.

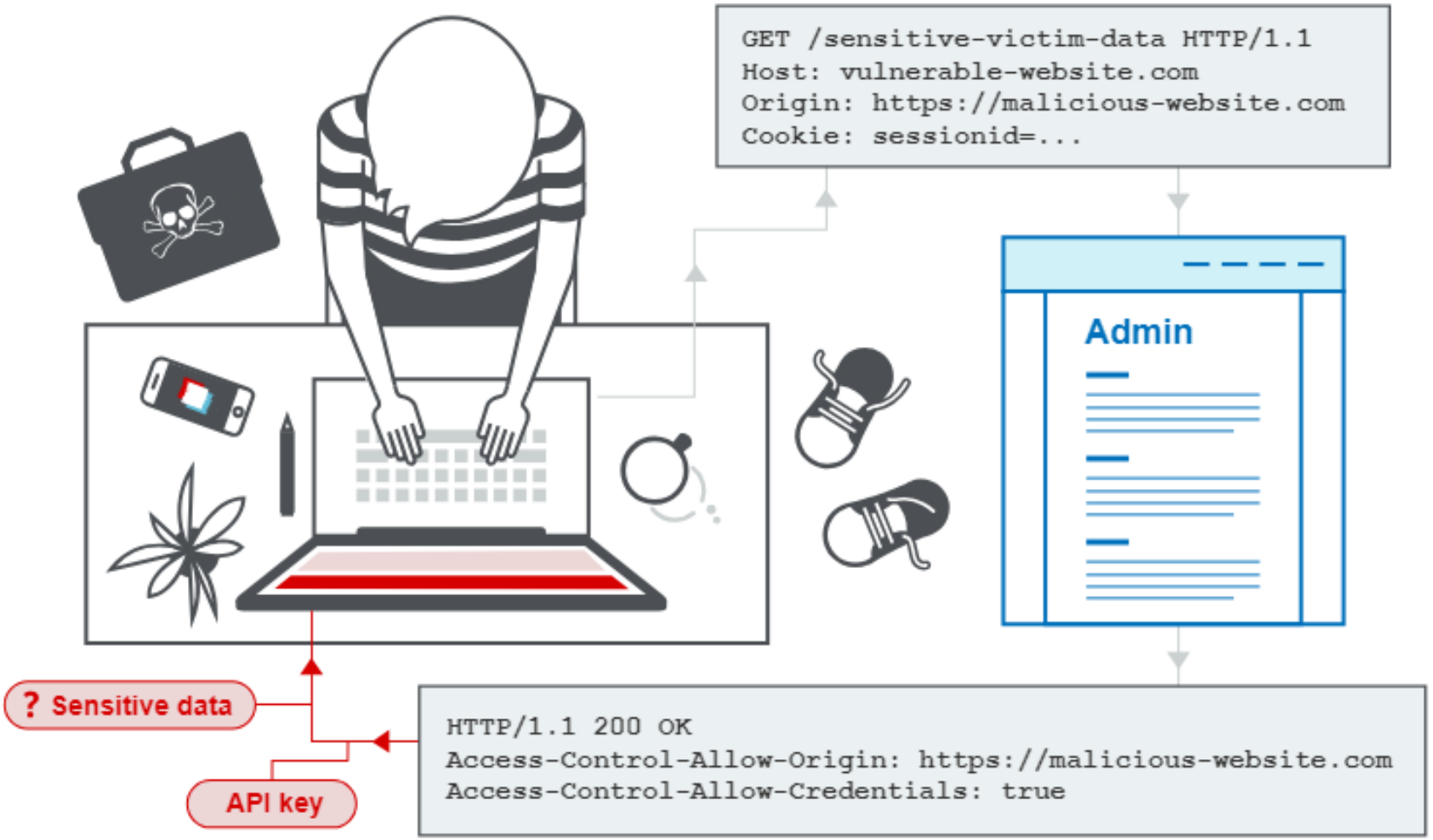
CORS is not a protection against cross-origin attacks such as cross-site request forgery (CSRF).

CSRF

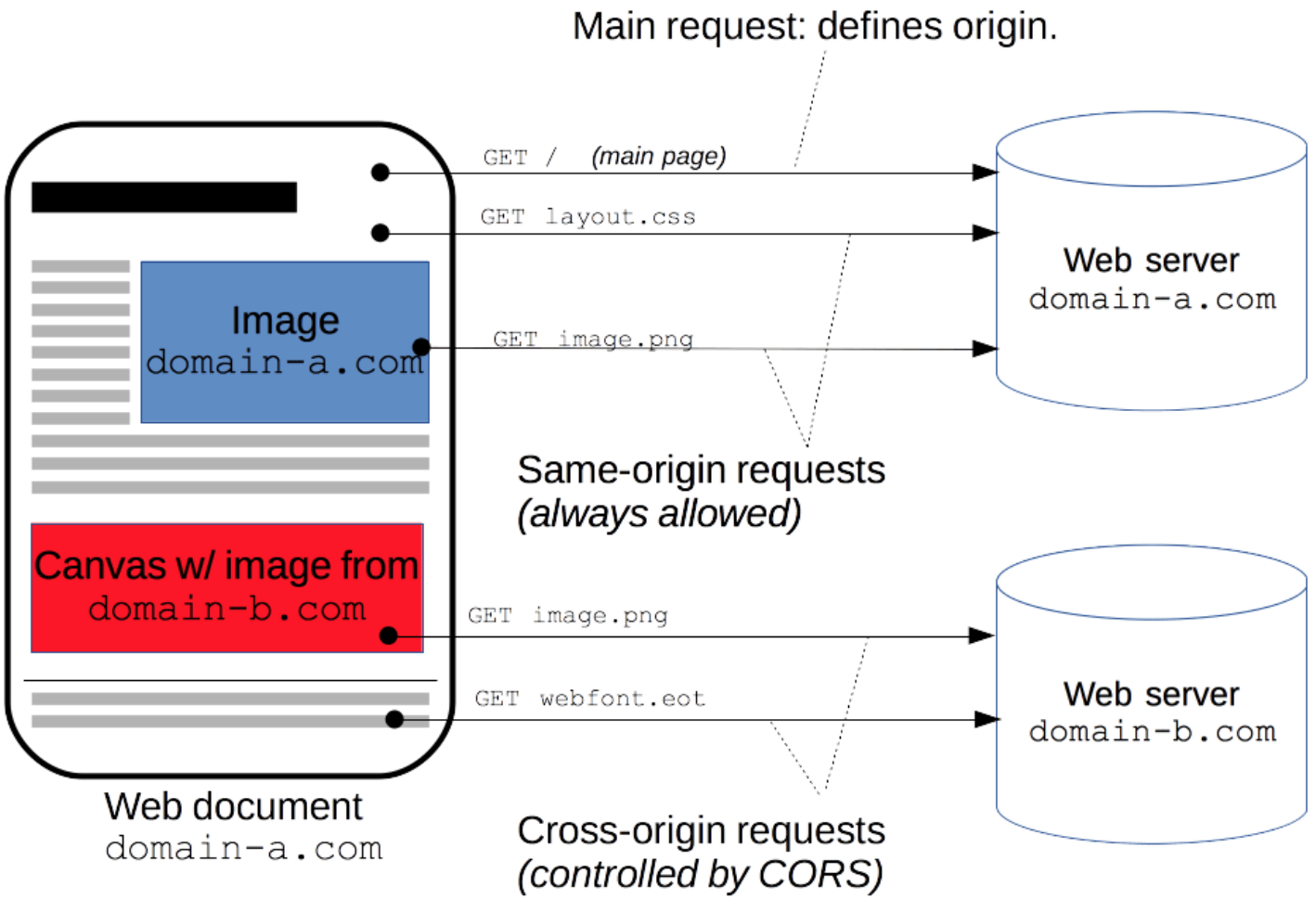
Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.



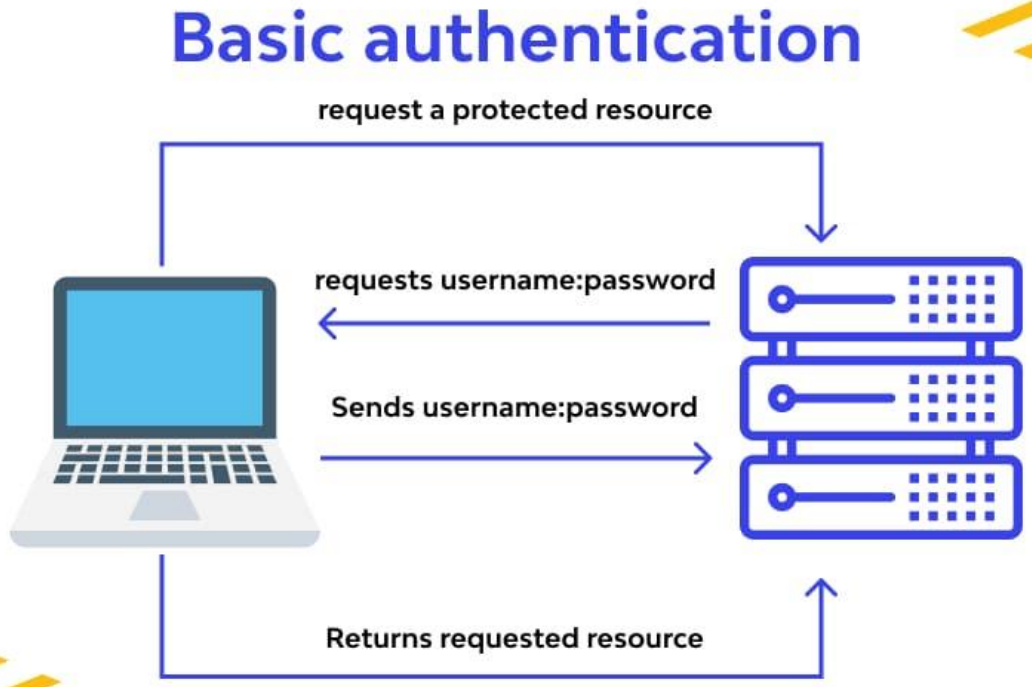
CORS


GET /sensitive-victim-data HTTP/1.1
Host: vulnerable-website.com
Origin: https://malicious-website.com
Cookie: sessionid=...
Admin
? Sensitive data
API key
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://malicious-website.com
Access-Control-Allow-Credentials: true

CORS


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
Canvas w/ image from
domain-b.com
Web document
domain-a.com
Web server
domain-a.com
Same-origin requests
(always allowed)
GET image.png
GET webfont.eot
Web server
domain-b.com
Cross-origin requests
(controlled by CORS)

Basic Authentication – meaning?



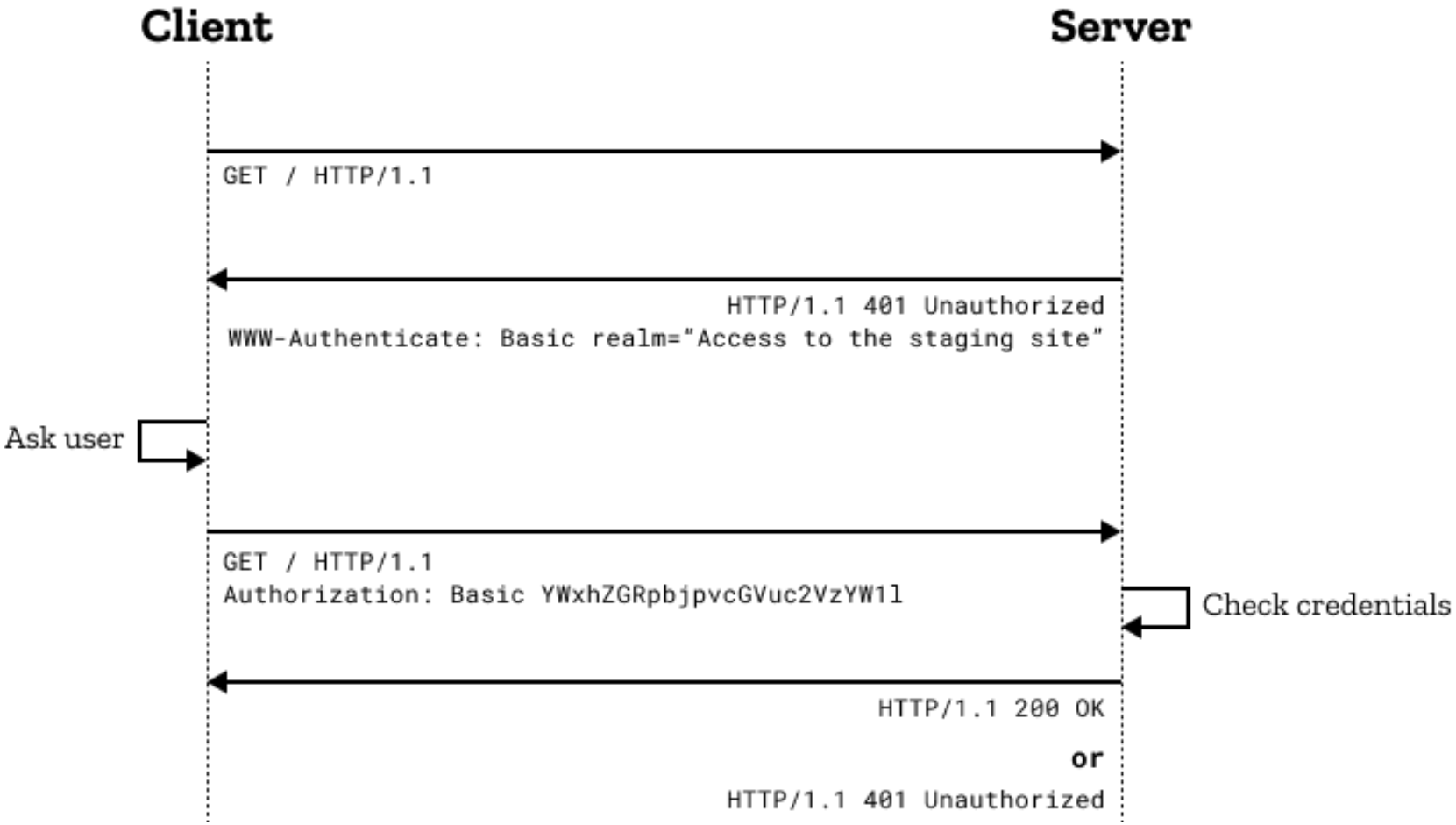
HTTP Authentication Framework

RFC 7235 defines the HTTP authentication framework, which can be used by a server to challenge a client request, and by a client to provide authentication information.

The challenge and response flow works like this:

1. The server responds to a client with a 401 (Unauthorized) response status and provides information on how to authorize with a WWW-Authenticate response header containing at least one challenge.
2. A client that wants to authenticate itself with the server can then do so by including an Authorization request header with the credentials.
3. Usually a client will present a password prompt to the user and will then issue the request including the correct Authorization header.

Generic HTTP Authentication



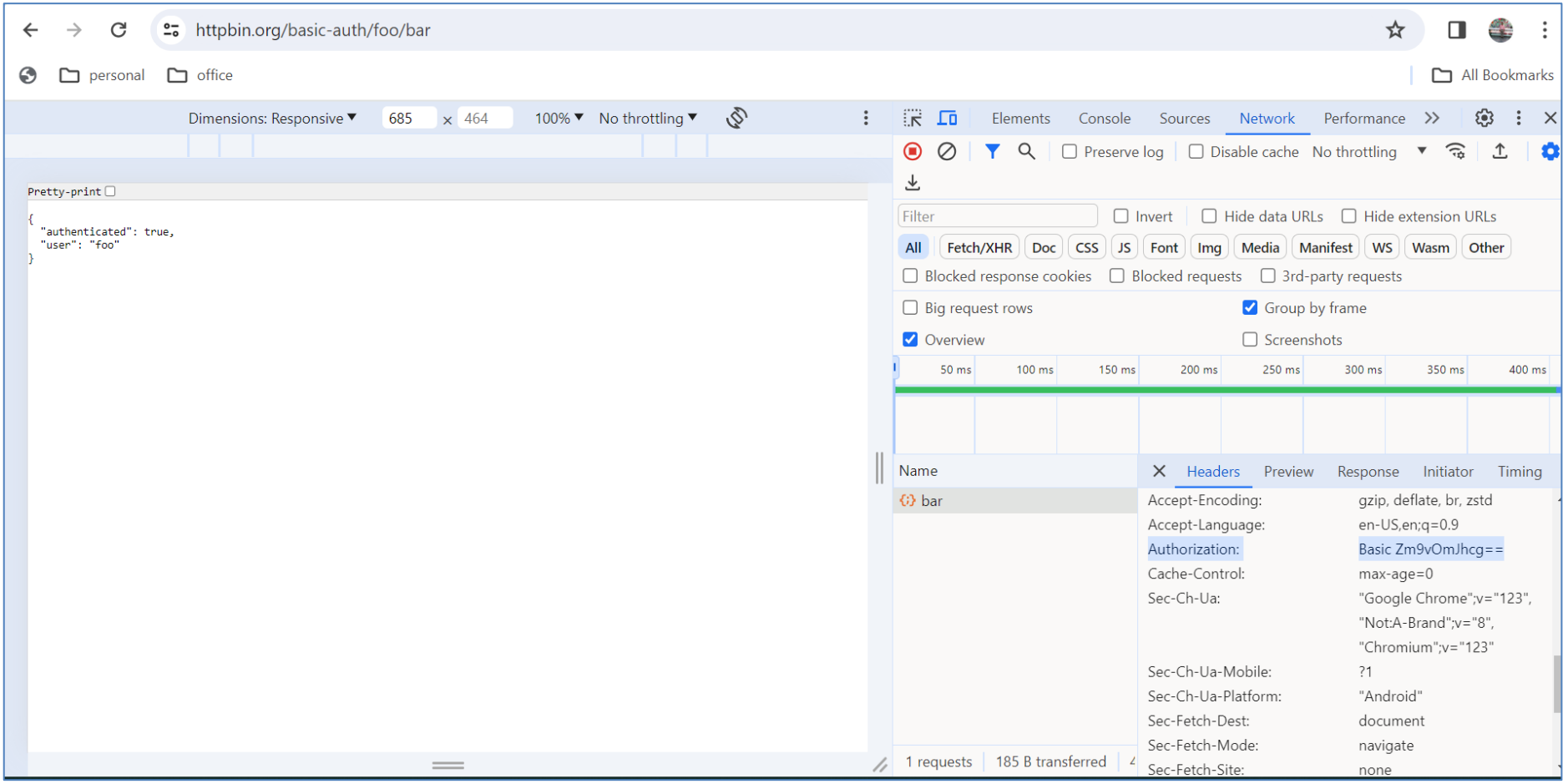
Basic Authentication

Basic authentication is a very simple authentication scheme that is built into the HTTP protocol. The client sends HTTP requests with the Authorization header that contains the Basic word followed by a space and a base64-encoded username:password string.

For example, a header containing the credentials demo / p@55w0rd would be encoded as:

Authorization: Basic ZGVtbzpwQDU1dzByZA==

Basic Authentication



How to define in Schema?

innovate achieve lead

```
securityDefinitions:
  basicAuth:
    type: basic

# To apply Basic auth to the whole API:
security:
  - basicAuth: []

paths:
  /something:
    get:
      # To apply Basic auth to an individual operation:
      security:
        - basicAuth: []
      responses:
        200:
          description: OK (successfully authenticated)
```

You can define 401 response

```yaml
paths:
  /something:
    get:
      ...
      responses:
        ...
        401:
          $ref: '#/responses/UnauthorizedError'
    post:
      ...
      responses:
        ...
        401:
          $ref: '#/responses/UnauthorizedError'
responses:
  UnauthorizedError:
    description: Authentication information is missing or invalid
    headers:
      WWW_Authenticate:
        type: string
```

SAML & OAuth

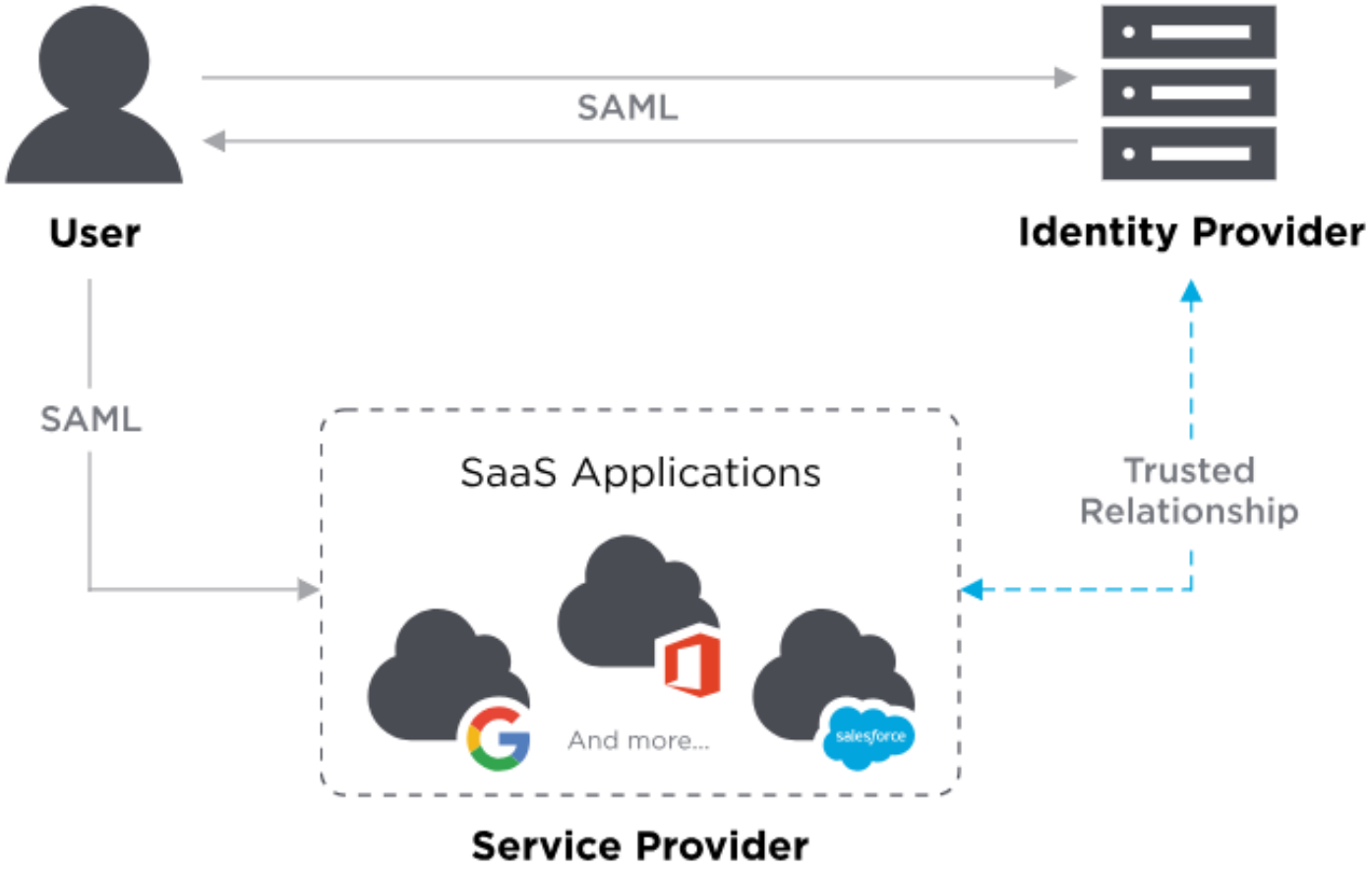
SAML

SAML stands for Security Assertion Markup Language. It's an open standard that allows secure logins across different websites and applications. Here's a breakdown of how it works:

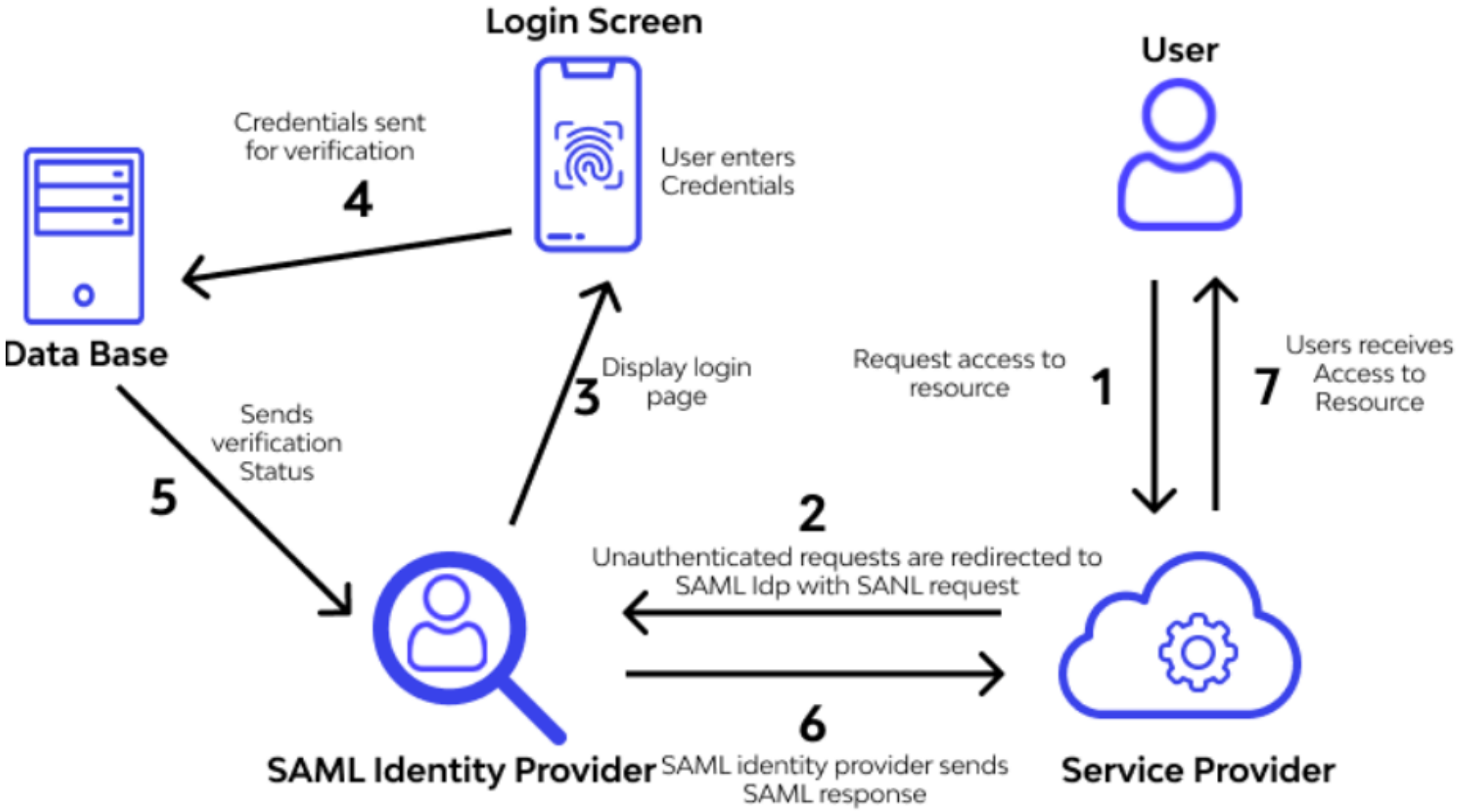
- **Imagine you have a central ID card**: This is like an Identity Provider (IdP) in SAML. It's a trusted service that verifies your identity (usually when you log in).
- **Websites you visit**: These are like Service Providers (SPs) in SAML. They rely on the IdP to confirm your identity instead of making you log in again.
- **The exchange of information**: When you try to access a website (SP), it sends a request to the IdP. The IdP checks your credentials and if valid, sends a secure message (assertion) back to the website saying you're good to go.

Essentially, SAML enables Single Sign-On (SSO) so you don't have to enter login credentials on multiple sites. It also improves security by centralizing authentication.

SAML


SAML
User
Identity Provider
SAML
SaaS Applications
Trusted
Relationship
And more...
salesforce
Service Provider

Authentication Process



SAML Example Steps

1 User logs into SSO

User tried to access a webpage

2

3 Service provider checks the users credentials with the identity provider

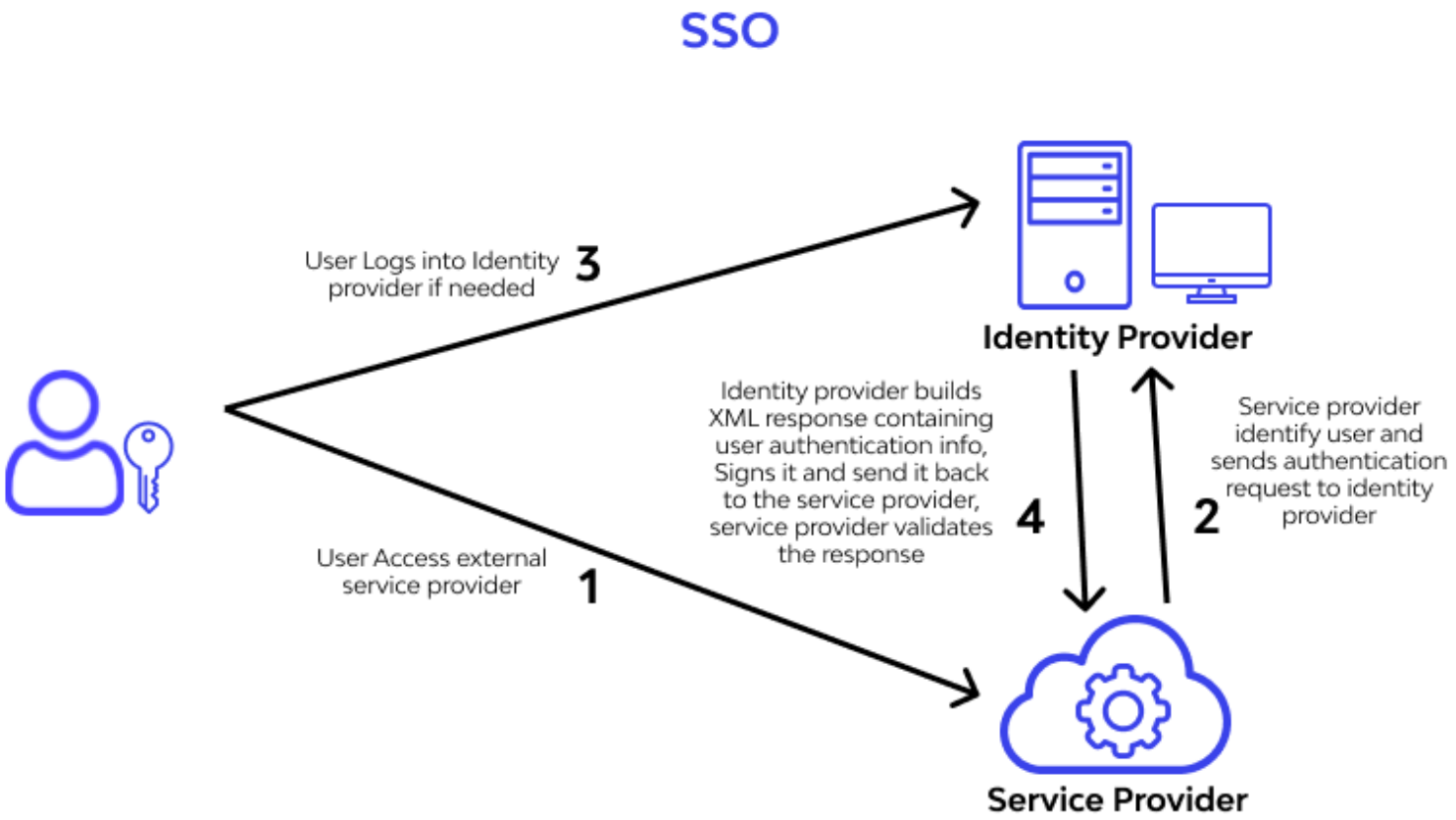
Identity provider sends authorization and authentication messages back to service provider

4

5 User can now log into the CRM

VARONIS

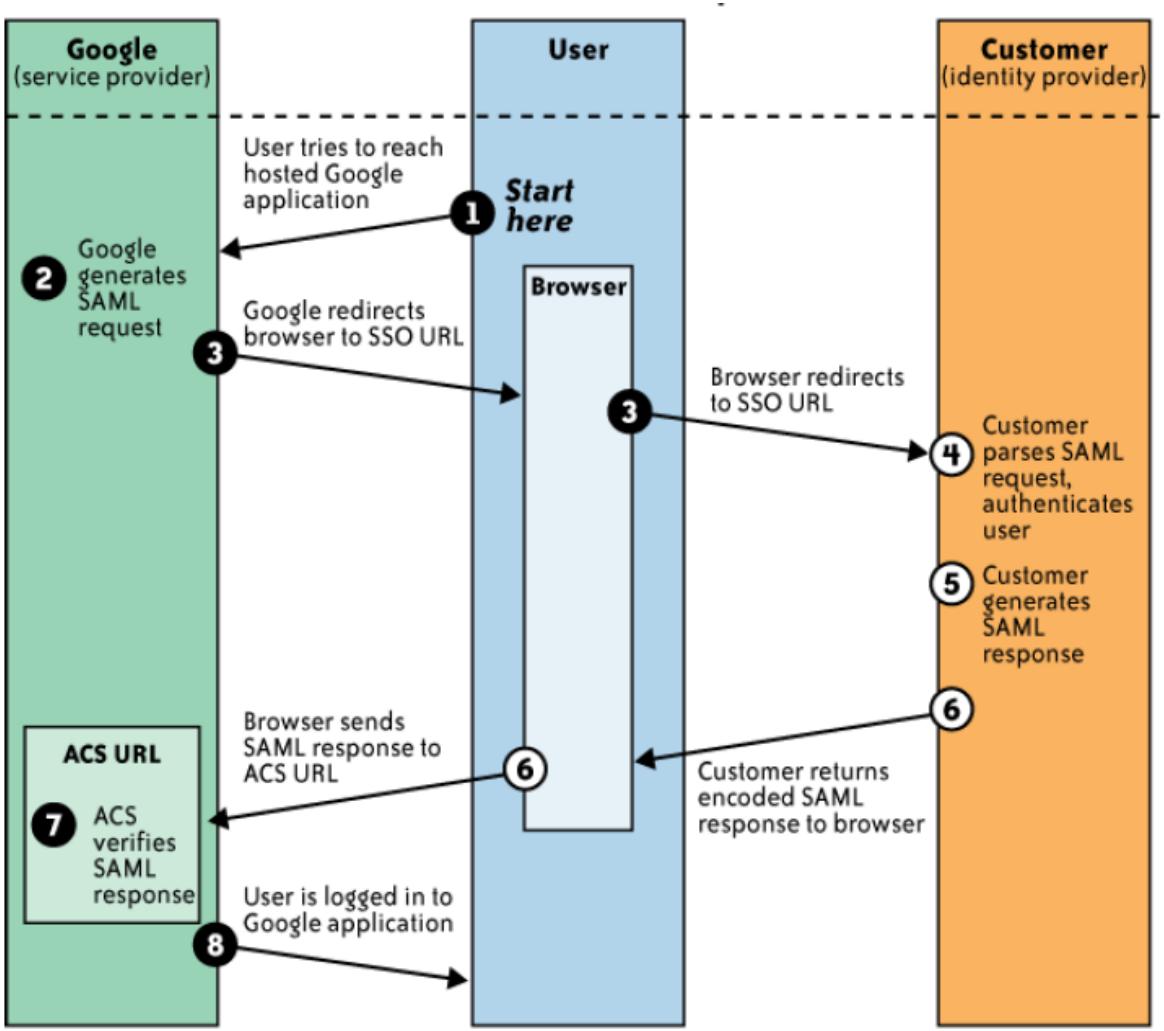
SAML and SSO


SSO
User Logs into Identity provider if needed
3
Identity Provider
Identity provider builds XML response containing user authentication info, Signs it and send it back to the service provider, service provider validates the response
4
Service provider identify user and sends authentication request to identity provider
2
User Access external service provider
1
Service Provider

SAML and SSO

	SAML	SSO
Purpose	SAML is the standard through which SPs and IdPs communicate with each other to verify credentials.	SSO is an authentication process intended to simplify access to multiple applications with a single set of credentials.
Features	SAML improves security by unburdening SPs from having to store login credentials. Instead, it places the responsibility on IdPs that specialize in such services.	SSO simplifies user experience (UX) by providing a singular access point for the multiple services and platforms users regularly access.
Use cases	SAML simplifies and controls authentication-related tasks. It enforces secure login protocols and manages authentication permissions across various platforms.	SAML facilitates SSO, the primary use of which enables integrated logins across an organization's multiple services.

SSO Transaction Steps Using SAML



SAML message


```
<?xml version="1.0" encoding="UTF-8"?>
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
                xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                ID="_abc123def456"
                Version="2.0"
                IssueInstant="2024-03-28T12:00:00Z"
                Destination="https://example.com/sso"
                InResponseTo="_xyz987uvw654">
    <saml:Issuer>https://idp.example.com</saml:Issuer>
    <samlp:Status>
        <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
    </samlp:Status>
    <saml:Assertion ID="_uvw987xyz654"
                    IssueInstant="2024-03-28T12:00:00Z"
                    Version="2.0">
        <saml:Issuer>https://idp.example.com</saml:Issuer>
        <saml:Subject>
            <saml:NameID>user@example.com</saml:NameID>
            <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
                <saml:SubjectConfirmationData NotOnOrAfter="2024-03-28T12:05:00Z"/>
            </saml:SubjectConfirmation>
        </saml:Subject>
        <saml:Conditions NotBefore="2024-03-28T12:00:00Z"
                         NotOnOrAfter="2024-03-28T12:05:00Z">
            <saml:AudienceRestriction>
                <saml:Audience>https://sp.example.com</saml:Audience>
            </saml:AudienceRestriction>
        </saml:Conditions>
        <saml:AuthnStatement AuthnInstant="2024-03-28T12:00:00Z">
            <saml:AuthnContext>
                <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextCl
                ...
```

OpenID

Imagine you have a master key that unlocks all the doors in a giant mall (the internet). Instead of needing a different key for each store (website), OpenID lets you use this master key from a trusted key keeper (like Google or Facebook) to verify who you are.

Here's how it works:

1. You visit a website.
2. The website says, "Hey, I don't know you, use your master key to prove yourself."
3. You choose your trusted key keeper (e.g., Google login).
4. You unlock the key with your password at the key keeper's place.
5. The key keeper sends a special signal back to the website saying, "This person is good to go!"

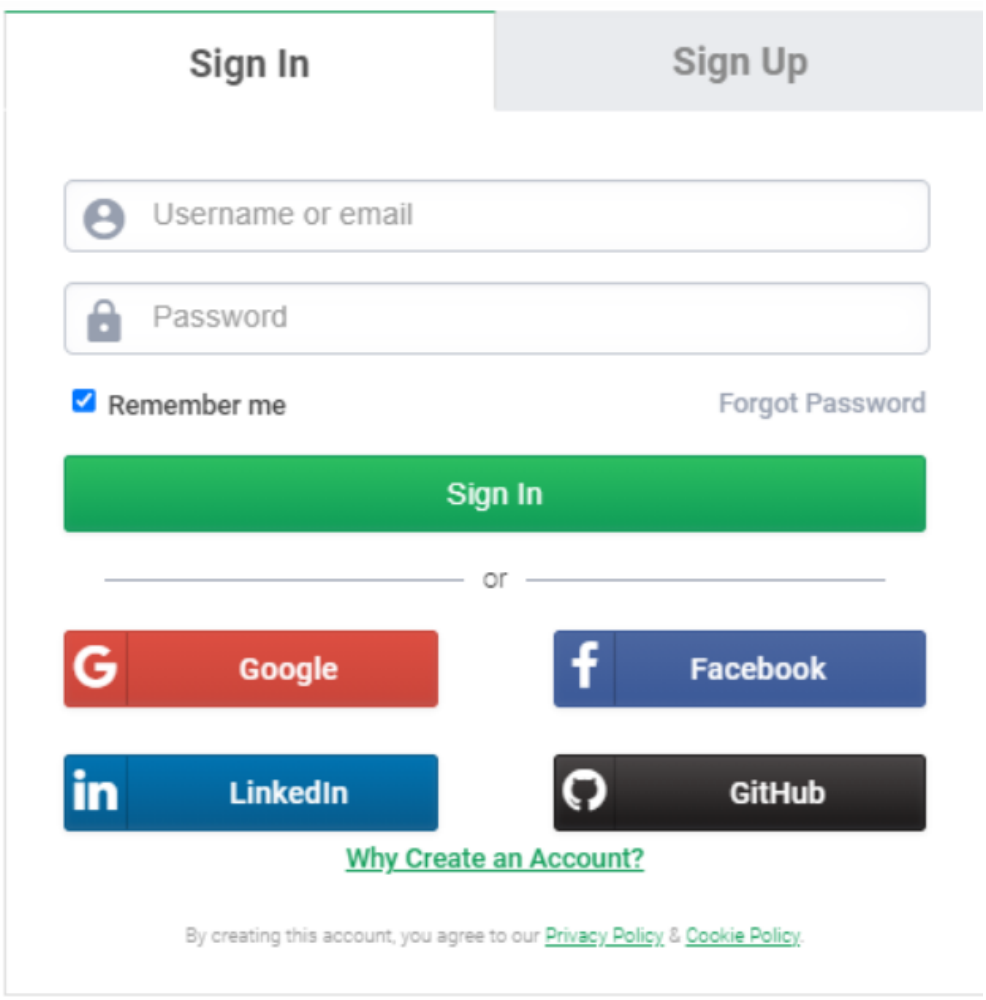
This way, you don't need separate logins for every website, and the website trusts you because the key keeper vouches for you. OpenID makes logging in easier and more secure.

OAuth

OAuth (think "Open Access")

- Deals with **authorization**, not identification.
- Imagine you (the user) own a house (your data) with many rooms (specific data points).
- OAuth lets you give an app (like a cleaning service) a temporary key (access token) to access a specific room (limited data) in your house.
- This key has an expiration date and can be revoked if needed.

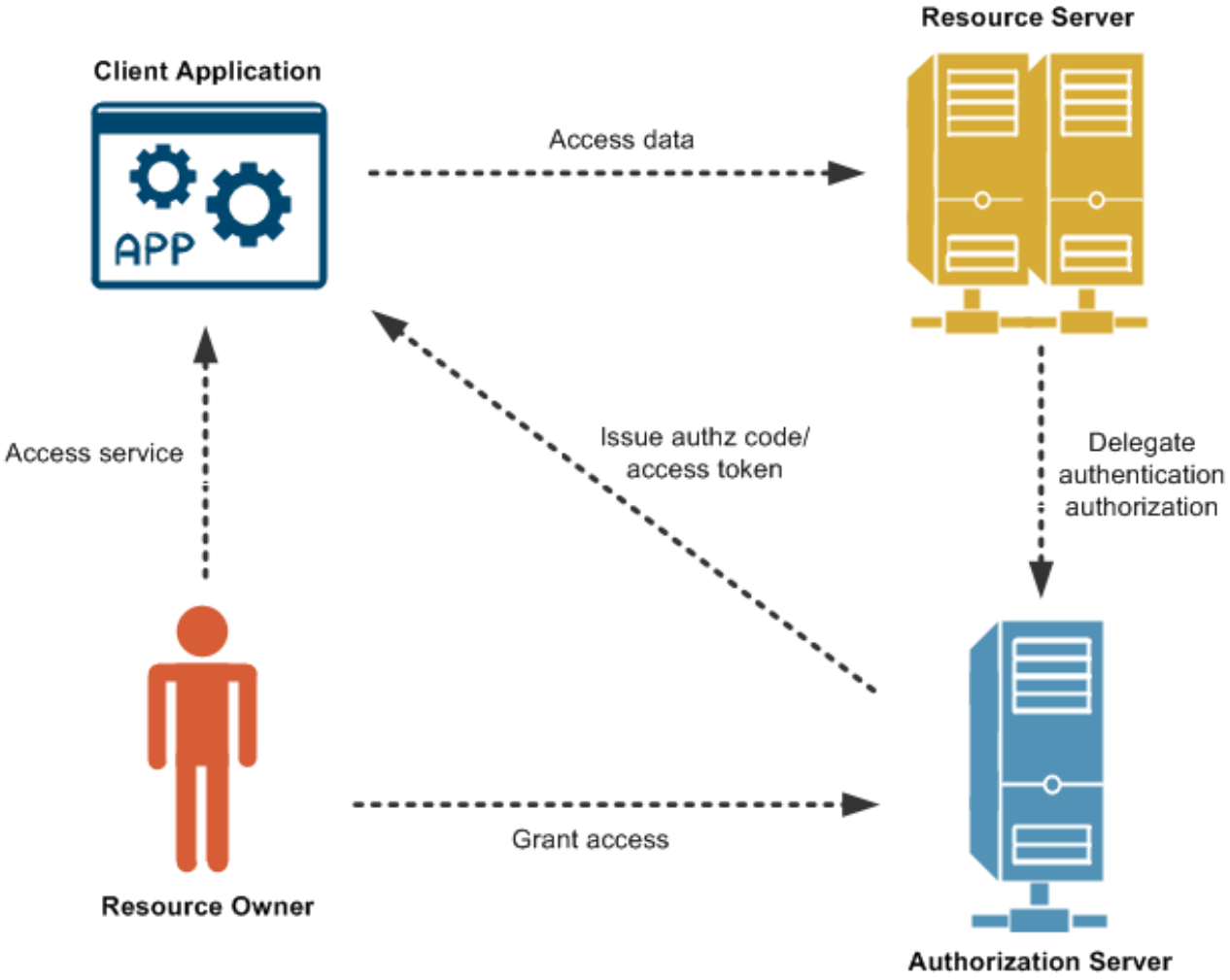
OAuth 2.0



OAuth is an open standard for **authorization** that enables client applications to access server resources on behalf of a specific Resource Owner. OAuth also enables Resource Owners (end users) to authorize limited third-party access to their server resources without sharing their credentials.

For example, a Gmail user could allow LinkedIn or Flickr to have access to their list of contacts without sharing their Gmail username and password.

OAuth 2.0

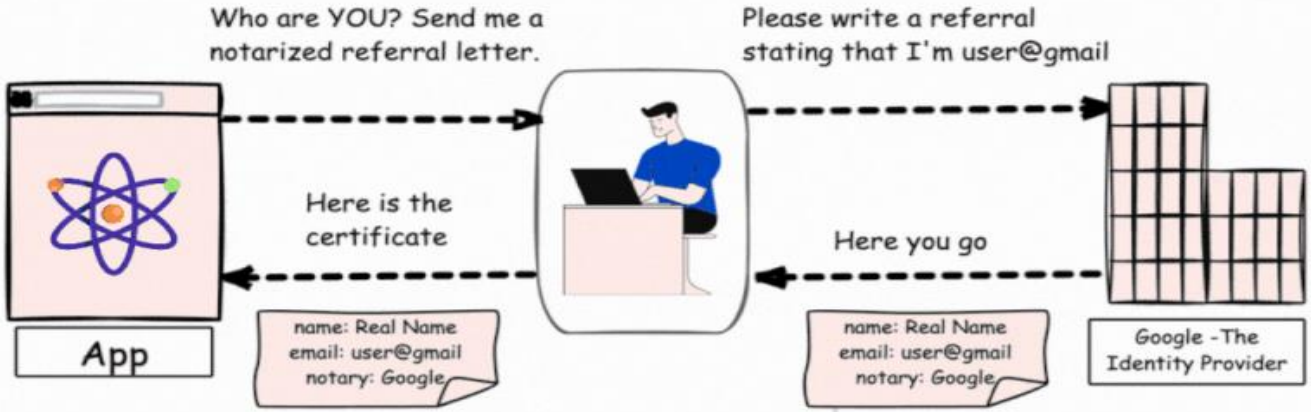

Resource Server
Client Application
APP
Access data
Access service
Issue authz code/
access token
Delegate
authentication
authorization
Grant access
Resource Owner
Authorization Server

OpenID vs. OAuth

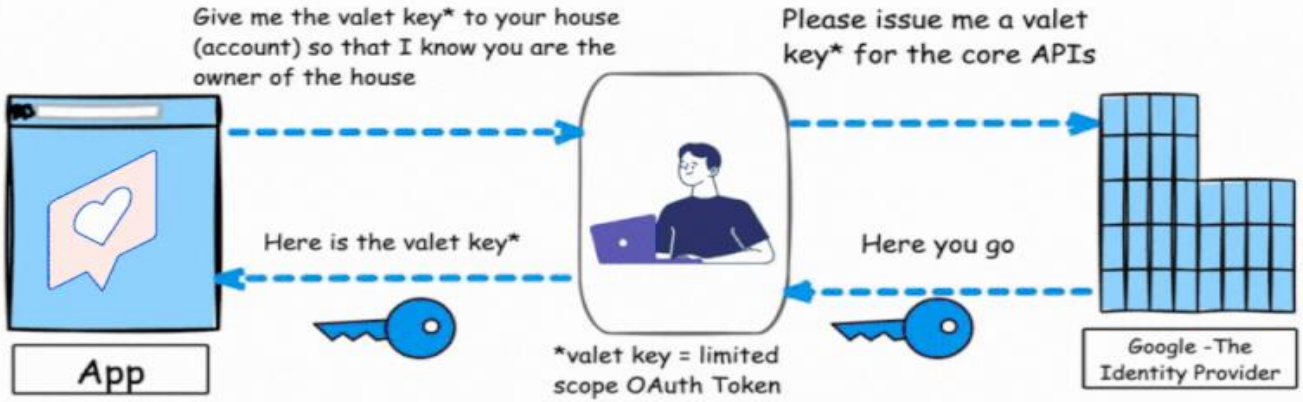
Reco SaaS Security

innovate achieve lead

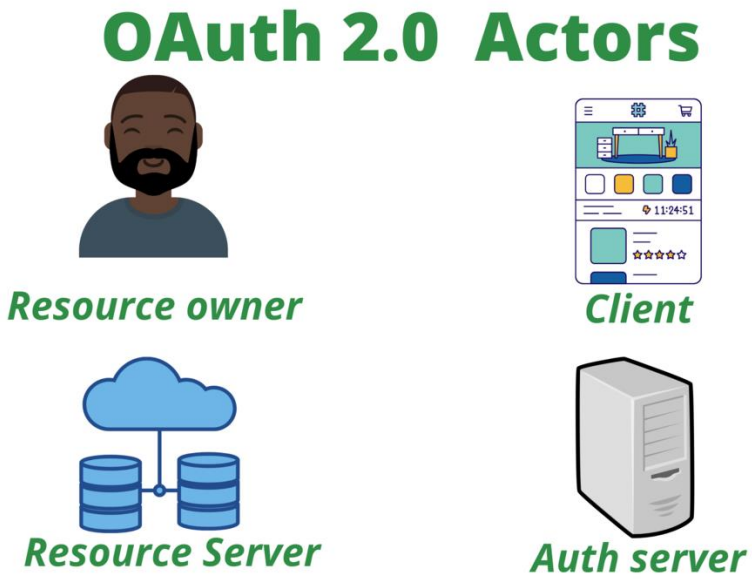
OpenID Authentication



Pseudo-Authentication using OAuth



OAuth 2.0 : Main Actors



- **Resources** are protected data that require OAuth to access them.
- **Resource Owner**: Owns the data in the resource server. An entity capable of granting access to protected data. For example, a user Google Drive account.
- **Resource Server**: The API which stores the data. For example, Google Photos or Google Drive.
- **Client**: It is a third-party application that wants to access your data, for example, a photo editor application.

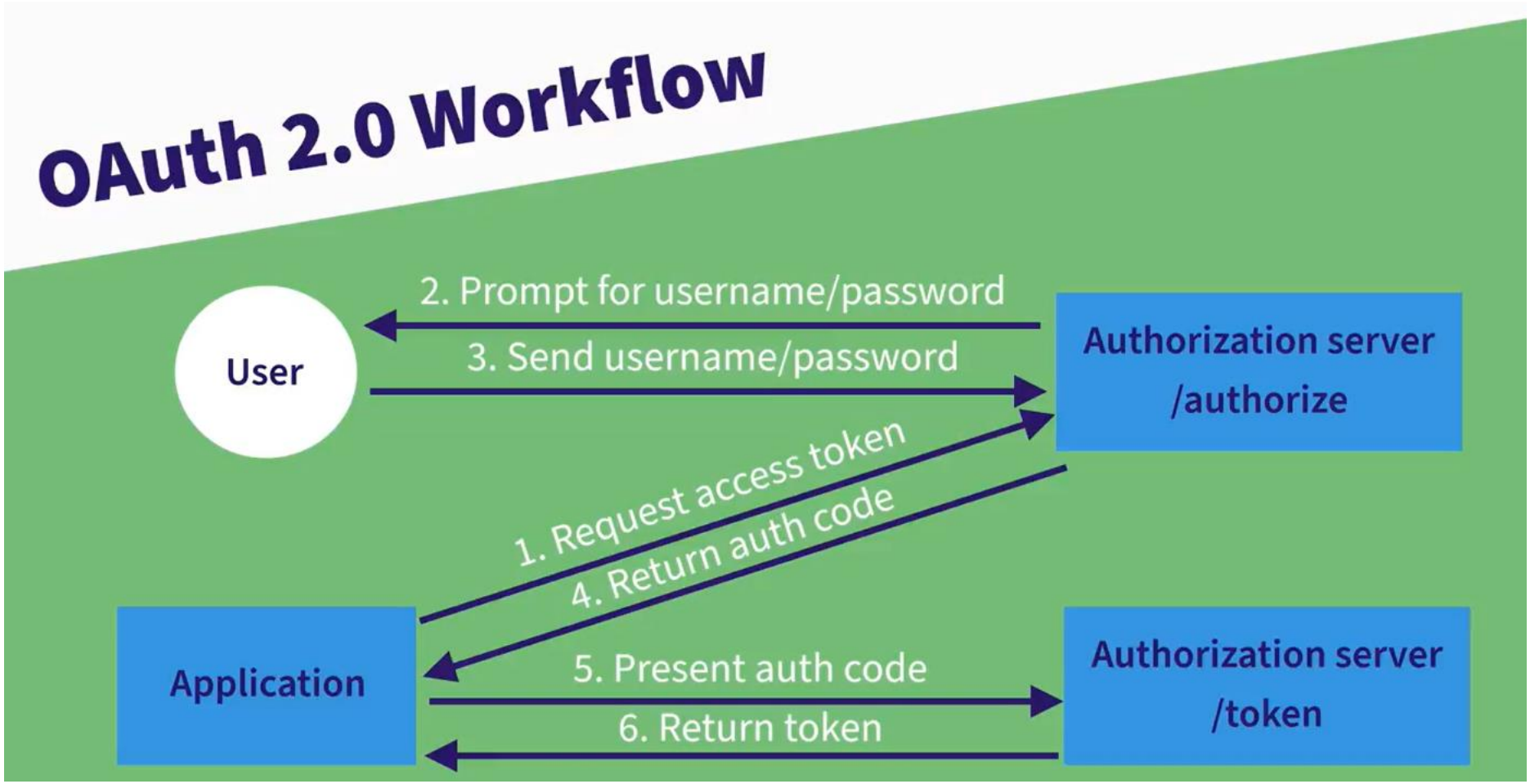
OAuth Main Actors

Principles of OAuth 2.0

OAuth 2.0 is an authorization protocol and NOT an authentication protocol. As such, it is designed primarily as a means of granting access to a set of resources, for example, remote APIs or user data.

OAuth 2.0 uses Access Tokens. An **Access Token** is a piece of data that represents the authorization to access resources on behalf of the end-user. OAuth 2.0 doesn't define a specific format for Access Tokens. However, in some contexts, the JSON Web Token (JWT) format is often used. This enables token issuers to include data in the token itself. Also, for security reasons, Access Tokens may have an expiration date.

OAuth Use Case



SAML vs OAuth

innovate achieve lead

SAML is a set of standards that have been defined to share information about who a user is, what his set of attributes are, and give you a way to grant/deny access to something or even request authentication.

OAuth is more about delegating access to something. You are basically allowing someone to "act" as you. Its most commonly used to grant access api's that can do something on your behalf.

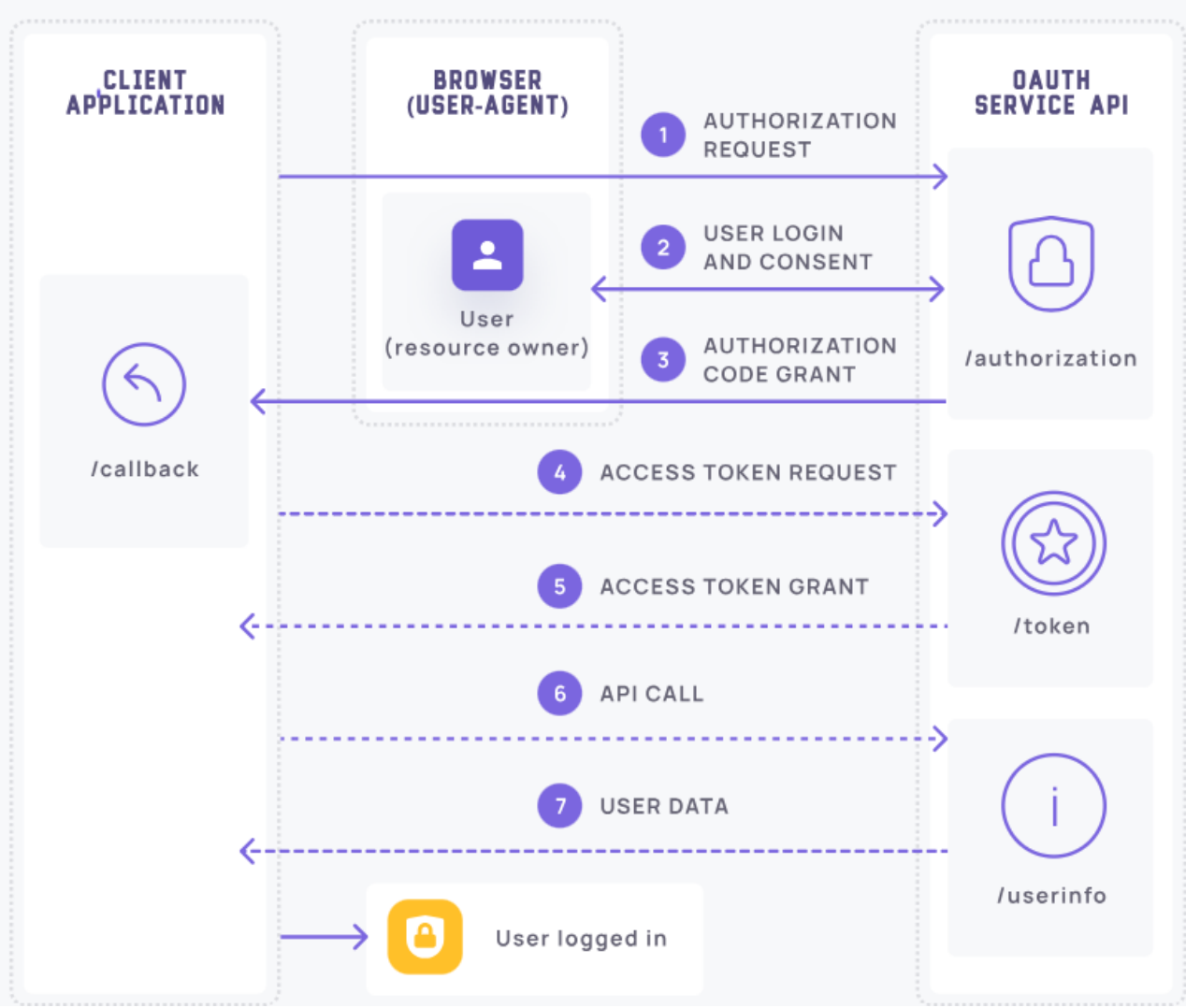
Comparison

PROTOCOL	OpenId	OAuth	SAML
What is it?	Open standard for authentication	Open standard for authorization	Open standard for authorization and authentication
History	Developed by the OpenId Foundation in 2014	Developed by Twitter and Google in 2006	Developed by OASIS in 2001
Current Version	OpenId Connect 1.0 released in 2014	2.0 released in 2012	2.0 released in 2005
Purpose	Provides an authentication layer over OAuth2.0	Enables delegated authorisation for internet resources	Allows 2 web entities to exchange authentication and authorization data
When to use	To authenticate users to your web or mobile app without requiring them to create an account	To provide temporary resource access to a 3rd party application on a legitimate user's behalf	To allow a use or corporate partner to use single sign-on to access a web service
Primary use case	SSO for consumer apps	API authorization	SSO for enterprise apps
Format	JSON	JSON	XML
Supported protocols	XRDS, HTTP	HTTP	HTTP, SOAP, and any protocols that can transport XML

Example Configuration

https://aaronparecki.com/oauth-2-simplified/

OAuth : Authorization Code Grant Flow



Authorization Code Grant Flow

The flow between the OAuth service and client application is kickstarted via a series of browser-based HTTP requests. Once the user consents to the access request, an authorization code is granted to the client application, which communicates with the OAuth service to get an “access token.” This token is very crucial, as it allows the making of API calls to fetch the required user data.

Following this exchange, all communications (server-to-server) are performed over safe back-channels, which are established during registration with the OAuth service (also, a client_secret is generated at this time, which the client application uses to authenticate itself while sending server-to-server requests). The end-user is not exposed to these communications in any way or form.

Implicit Grant Type

The Implicit Grant Type was designed for applications where the token is returned directly to the browser without the need for an intermediate server step. It's considered a simpler flow than the Authorization Code, but at the cost of lesser security.

The Implicit Grant Flow:

1. The client sends the user to the authorization server with a request for authorization.
2. The user authenticates with the authorization server and gives consent.
3. Instead of returning an authorization code, the authorization server directly sends the access token as a fragment in the URL to the client's redirect URI.
4. The application extracts the token from the URL and uses it to access the protected resources.

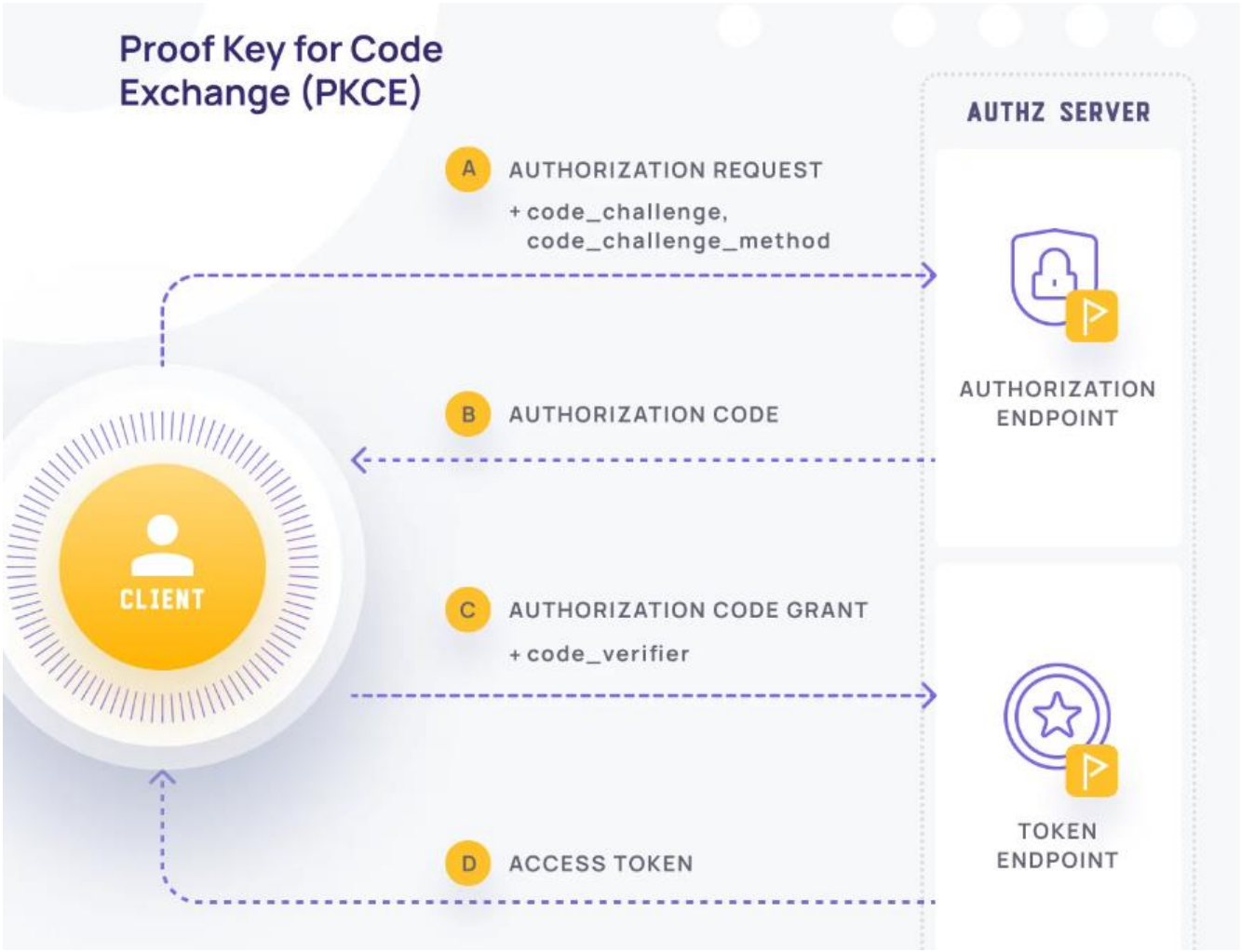
This grant type is less recommended nowadays due to potential security vulnerabilities, especially in the modern web application environment. The absence of client authentication and the exposure of the token in the URL are primary concerns.

Proof Key for Code Exchange (PKCE)

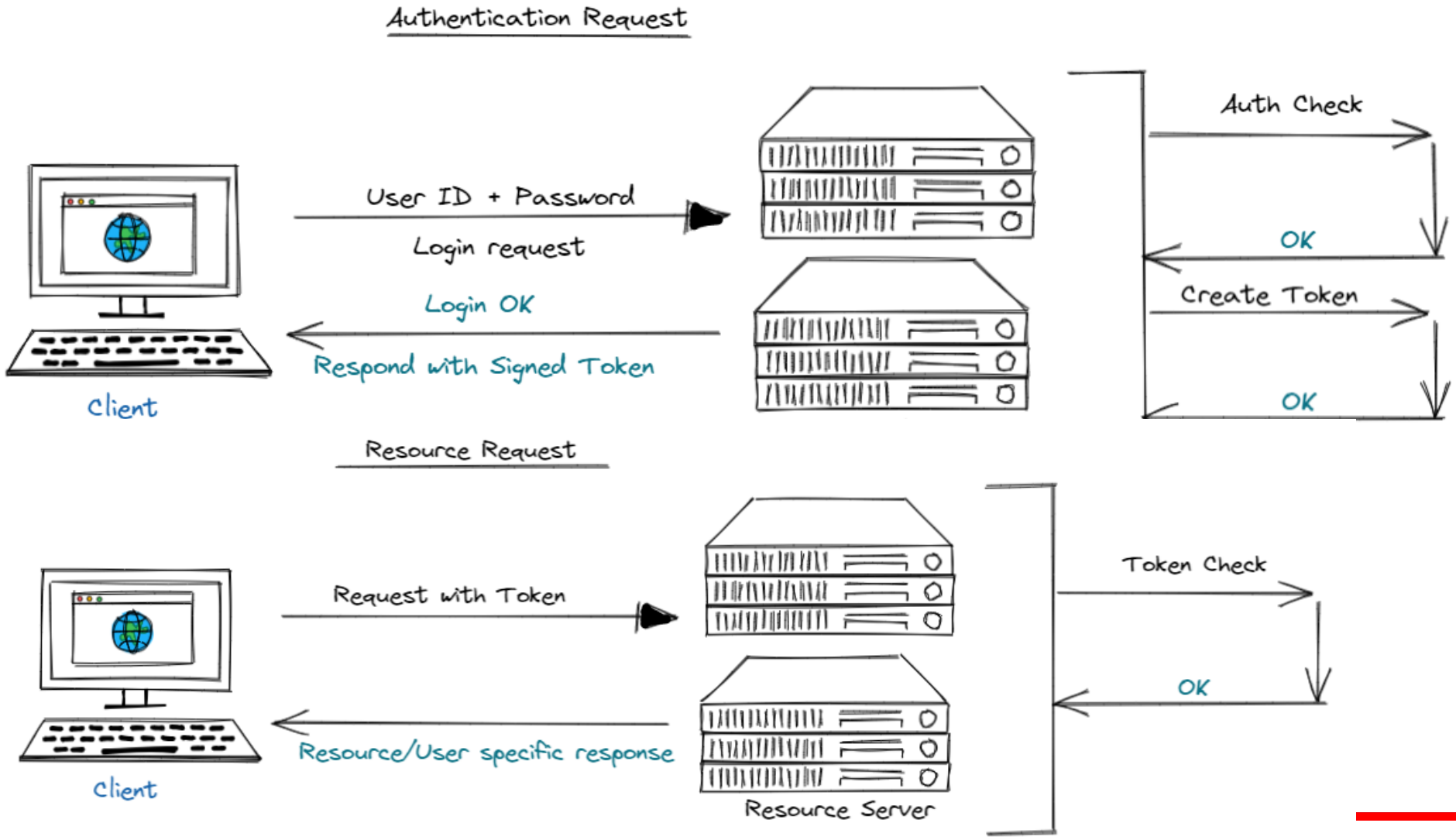
Proof Key for Code Exchange is a security-centric OAuth grant type. The main concept behind PKCE is proof of possession.

This basically means that the client app needs to prove to the authorization server that the authorization code is authentic, before getting an access token from it. The PKCE flow includes a code verifier and a code challenge, along with a code challenge method.

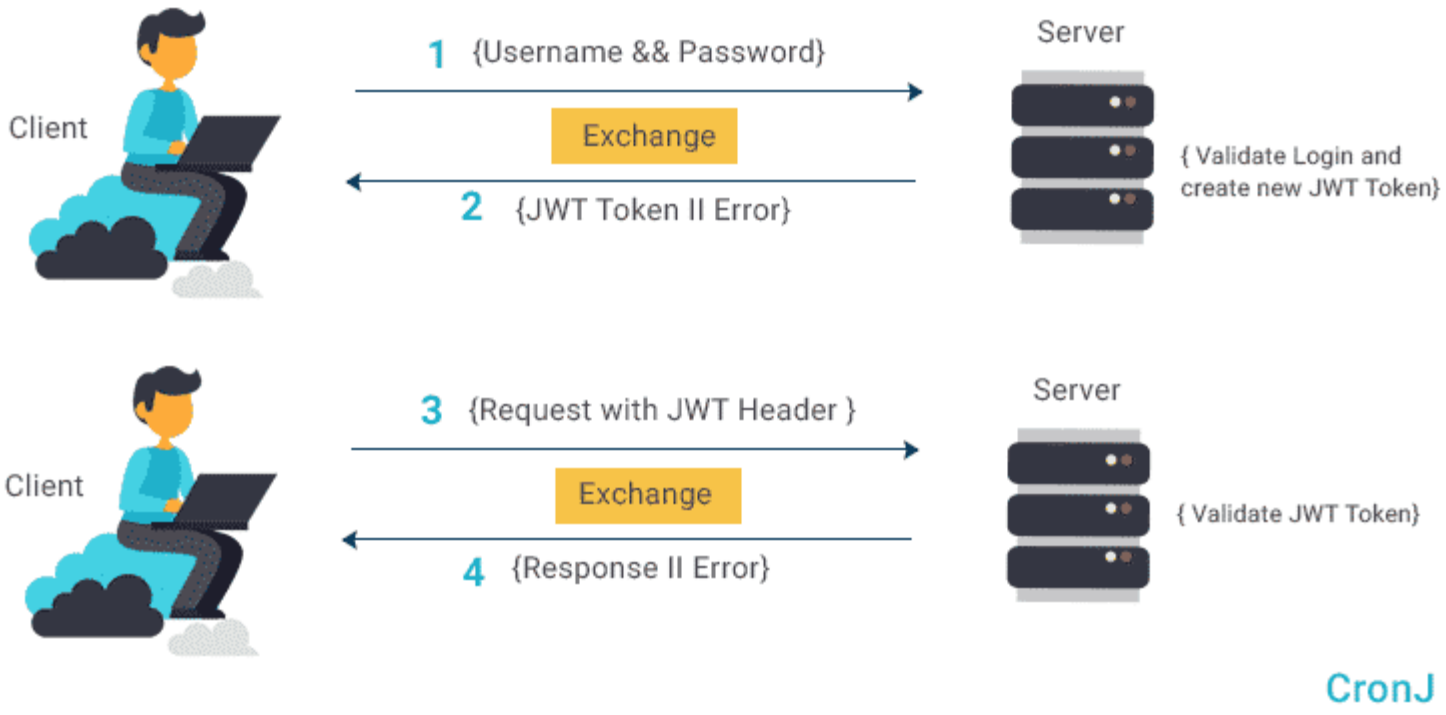
PKCE flow


Proof Key for Code Exchange (PKCE)
AUTHZ SERVER
A AUTHORIZATION REQUEST
+ code_challenge, code_challenge_method
AUTHORIZATION ENDPOINT
B AUTHORIZATION CODE
CLIENT
C AUTHORIZATION CODE GRANT
+ code_verifier
TOKEN ENDPOINT
D ACCESS TOKEN

Token based Authentication Flow



Resource Request



Question

innovate achieve lead

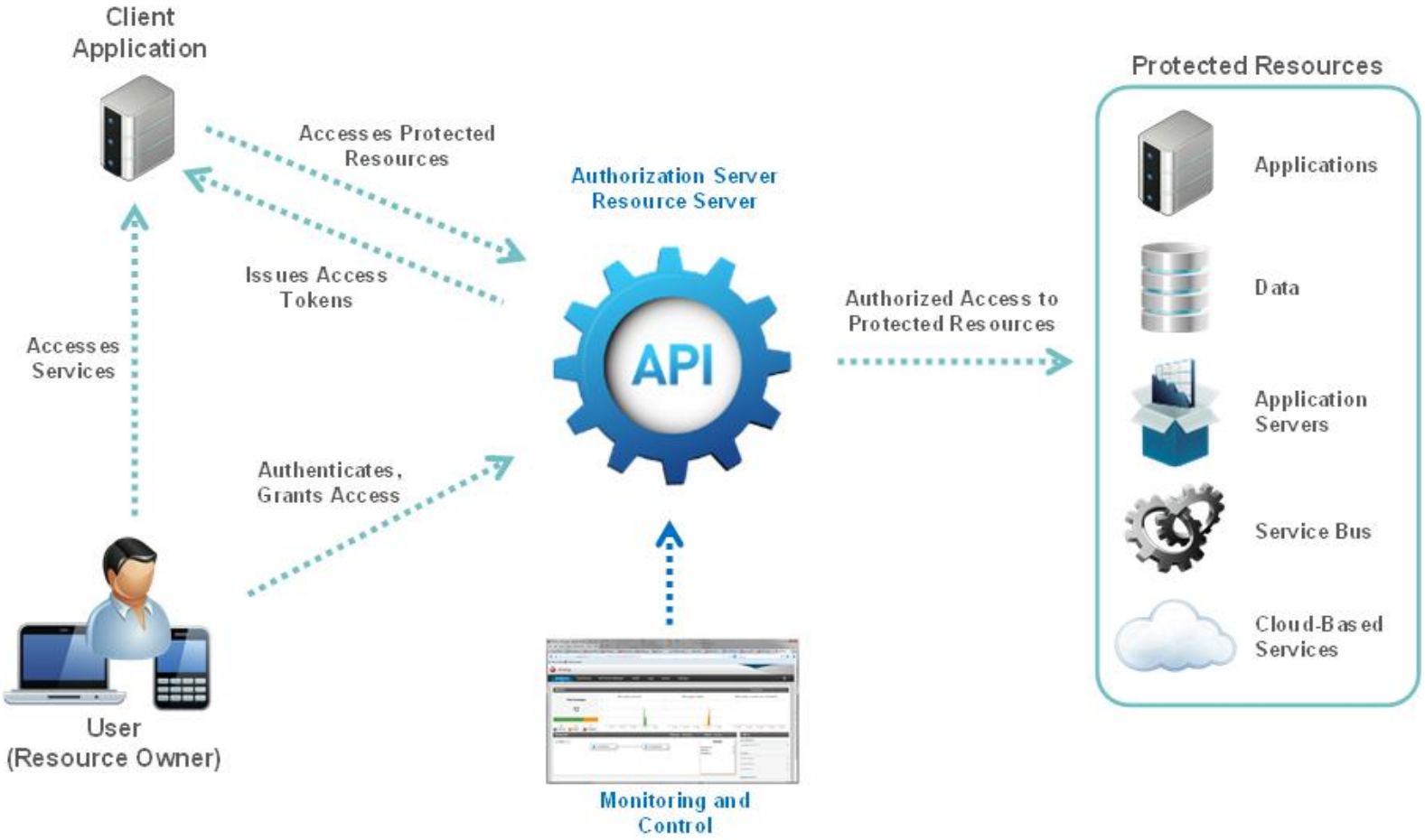
Why use OAuth to protect your APIs?

API Gateway OAuth Features

The API Gateway uses the following definitions of basic OAuth 2.0 terms:

- **Resource Owner**: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user.
- **Resource Server**: The server hosting the protected resources, and which is capable of accepting and responding to protected resource requests using access tokens. In this case, the API Gateway acts as a gateway implementing the Resource Server that sits in front of the protected resources.
- **Client Application**: A client application making protected requests on behalf of the resource owner and with its authorization.
- **Authorization Server**: The server issuing access tokens to the client application after successfully authenticating the Resource Owner and obtaining authorization. In this case, the API Gateway acts both as the Authorization Server and as the Resource Server.
- **Scope**: Used to control access to the Resource Owner's data when requested by a client application. You can validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. An example scope is https://localhost:8090/auth/userinfo.email.

Answer – API Gateway



OAuth 2 – Authentication Flows

The API Gateway supports the following authentication flows:

- **OAuth 2.0 Authorization Code Grant (Web Server)**:
- The Web server authentication flow is used by applications that are hosted on a secure server. A critical aspect of the Web server flow is that the server must be able to protect the issued client application's secret.
- **OAuth 2.0 Implicit Grant (User-Agent)**:
- The user-agent authentication flow is used by client applications residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript or Flash. These client applications cannot keep the client application secret confidential.
- **OAuth 2.0 Resource Owner Password Credentials**:
- This username-password authentication flow can be used when the client application already has the Resource Owner's credentials.
- **OAuth 2.0 Client Credentials**:
- This username-password flow is used when the client application needs to directly access its own resources on the Resource Server. Only the client application's credentials are used in this flow. The Resource Owner's credentials are not required.
- **OAuth 2.0 JWT**:
- This flow is similar to OAuth 2.0 Client Credentials. A JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.

Question

What problems these protocols solve?

(OAuth2.0, SAML, and OpenID)

Answer

These protocols all deal with authentication and authorization, but in different ways:

- **OAuth2.0** focuses on **authorization**. It lets users grant access to their data on one platform (like Facebook) to another platform (like a photo editing app) without sharing their password.
- **SAML (Security Assertion Markup Language)** is a protocol for **Single Sign-On (SSO)**. This means users can log in once to a central system and access multiple applications without needing to re-enter credentials for each one. SAML is commonly used within organizations for enterprise applications.
- **OpenID Connect (OIDC)** builds on OAuth2.0 to add an **ID token**. This token contains information about the user, like their name and email, which can be used for login purposes (similar to SSO). OIDC is popular for web and mobile applications.

Question

When would you choose one over the others?

(OAuth2.0, SAML, and OpenID)

Answer

innovate achieve lead

- Choose **OAuth2.0** if you need a user to grant access to specific data on their account to another application.
- Choose **SAML** if you want a secure SSO solution for users within your organization to access various internal applications.
- Choose **OIDC** if you're developing a web or mobile app that needs both authorization and basic user information for login or profile purposes.

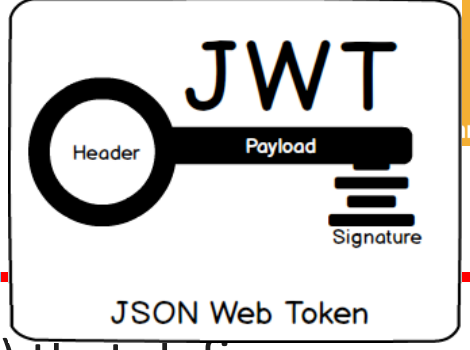
innovate achieve lead

BITS Pilani

Pilani Campus

JWT

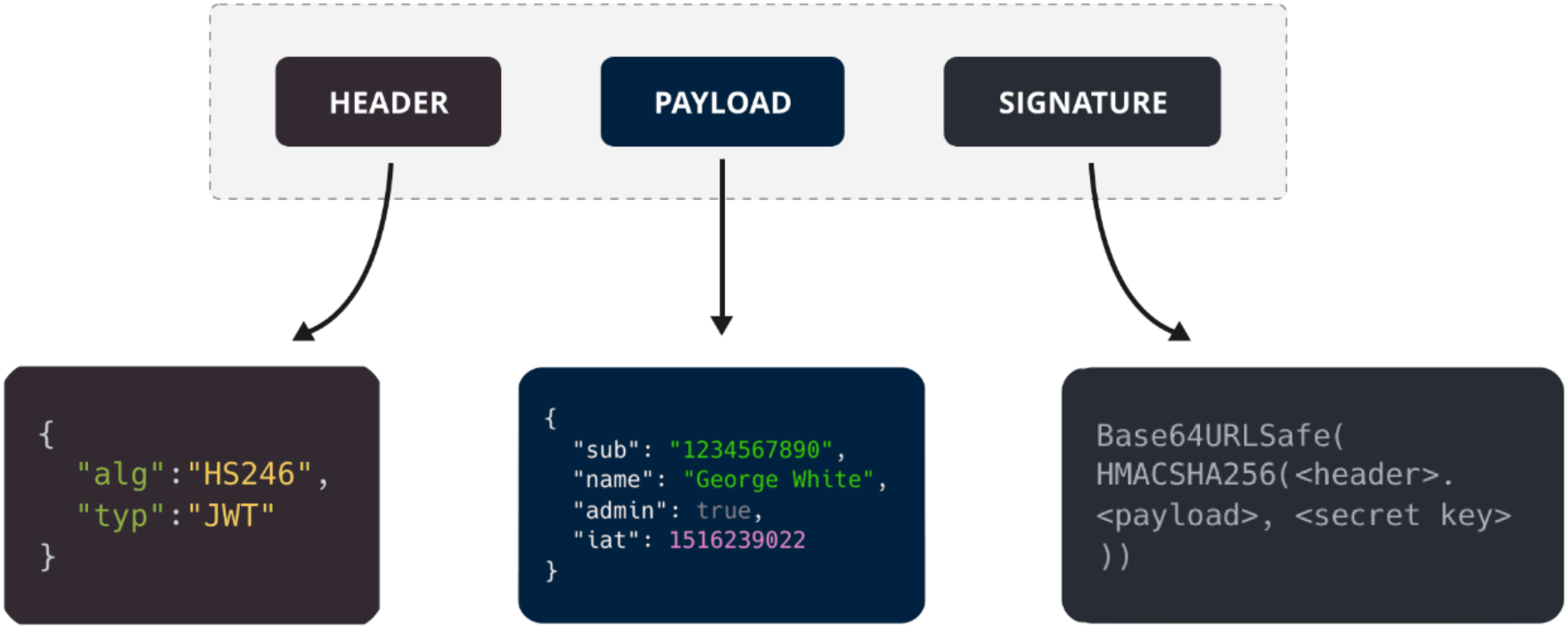
JSON Web Tokens



JWT is an open standard (RFC 7519) that defines a way to securely transmit information between two parties. This information is encoded as a JSON object with three parts:

1. **Header:** This contains information about the token itself, like the signing algorithm used.
2. **Payload:** This is the most important part. It carries the claims, which are essentially statements about the user or some other entity. These claims can include user ID, name, expiration time, etc.
3. **Signature:** This is generated using the header and payload, along with a secret key. It ensures the integrity of the data and helps verify that the sender is who they say they are.

JWT Structure

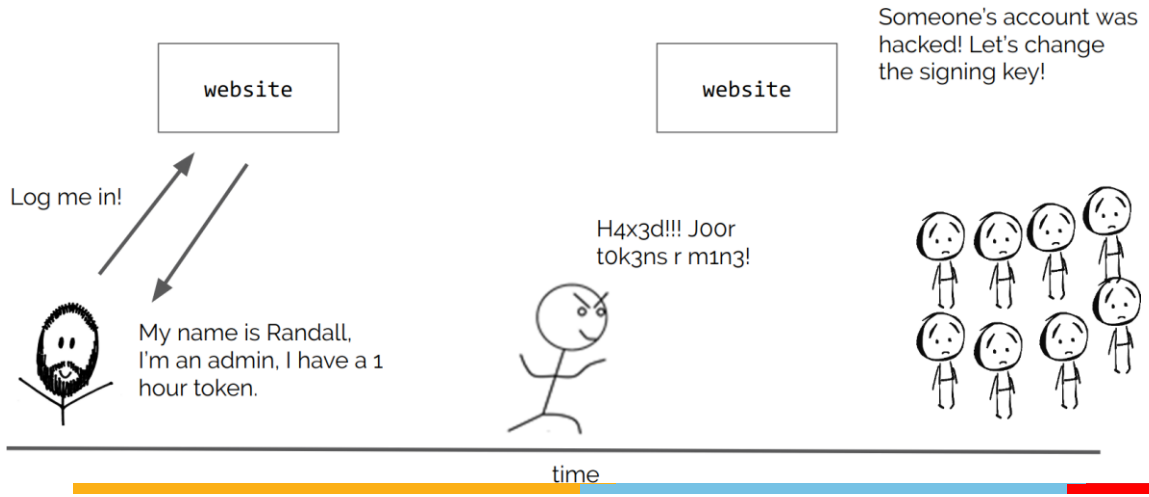


What's Good

- **Secure**: JWTs are digitally signed using either a secret (HMAC) or a public/private key pair (RSA or ECDSA) which safeguards them from being modified by the client or an attacker.
- **Stored only on the client**: You generate JWTs on the server and send them to the client. The client then submits the JWT with every request. This saves database space.
- **Efficient / Stateless**: It's quick to verify a JWT since it doesn't require a database lookup. This is especially useful in large distributed systems.

Drawback

- **Non-revocable**: Due to their self-contained nature and stateless verification process, it can be difficult to revoke a JWT before it expires naturally. Therefore, actions like banning a user immediately cannot be implemented easily. That being said, there is a way to maintain JWT deny / black list, and through that, we can revoke them immediately.
- **Dependent on one secret key**: The creation of a JWT depends on one secret key. If that key is compromised, the attacker can fabricate their own JWT which the API layer will accept. This in turn implies that if the secret key is compromised, the attacker can spoof any user's identity. We can reduce this risk by changing the secret key from time to time.



How does JWT works with OAuth 2.0

- Typically, after a successful login using OAuth2.0, an authorization server will issue a JWT containing user claims.
- The client application (like your mobile app) will receive this JWT and store it securely.
- With subsequent requests to access resources, the client will include the JWT in the authorization header.
- The resource server will then verify the JWT's signature and expiration time before granting access.

Flow


USER
1) Accesses...
CLIENT APP
(Front-end application)
2) Client prompts
for credentials
3) Credentials
Log in
Username
Password
Log in
7) JWT token
sent back to client
4) Credentials
passed to API
USER DATABASE
WEB API
your API: requires users to
be authenticated in order to
perform some actions

Summary

	JWT	OAuth	SAML
Purpose	Token-based authentication mechanism for transmitting claims	Protocol for authorization and authentication in web and mobile apps	Protocol for exchanging authentication and authorization data between parties
Centralized Server	None	Authorization server	Identity provider
Used For	Authentication and authorization	Authorization, not authentication	Authentication and authorization
Applications	Single-page apps, mobile apps	Apps that rely on external APIs or services	Enterprise environments
Relationship	Directly between parties	Between third-party apps and resource owners	Between identity providers and service providers
Authentication	Yes	No	Yes
Authorization	Yes	Yes	Yes