

How Authentication / Authorisation Works

3 steps:

- Get magic link
- Login via magic link
- Do whatever you need to do

See slides on Behind-The-Scenes
and Security Concerns for details

Step 1: Get magic link

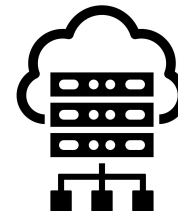
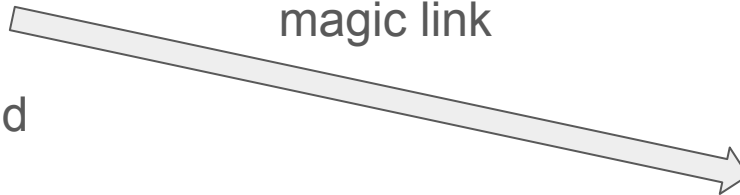
[Skip to behind-the-scenes](#)

1a: type email address in



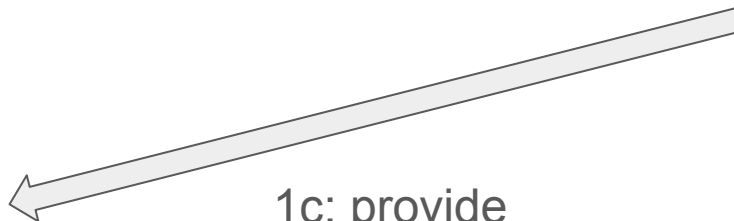
Grantbii frontend
(browser)

1b: request
magic link



Grantbii backend
(server)

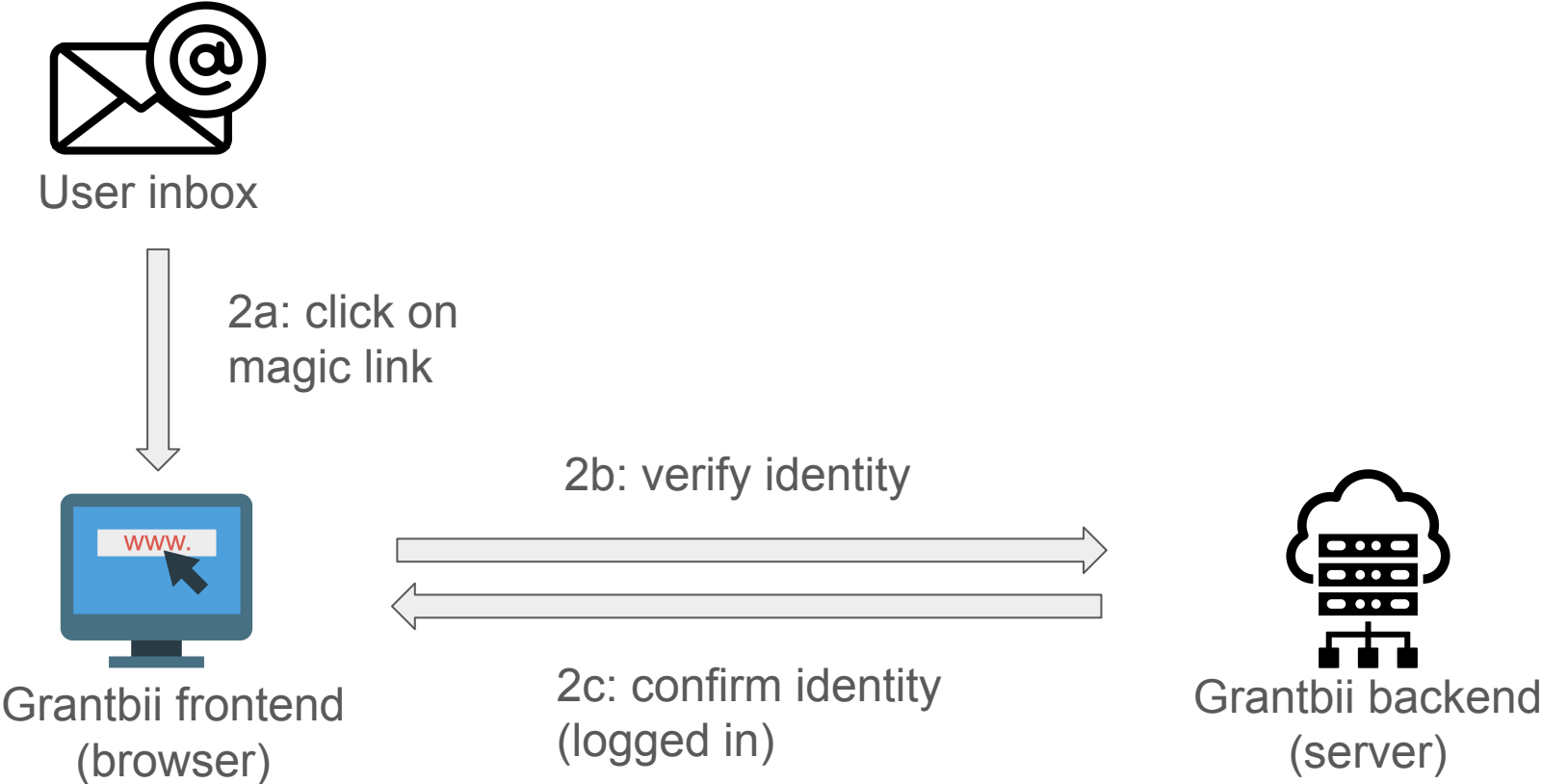
1c: provide
magic link



User inbox

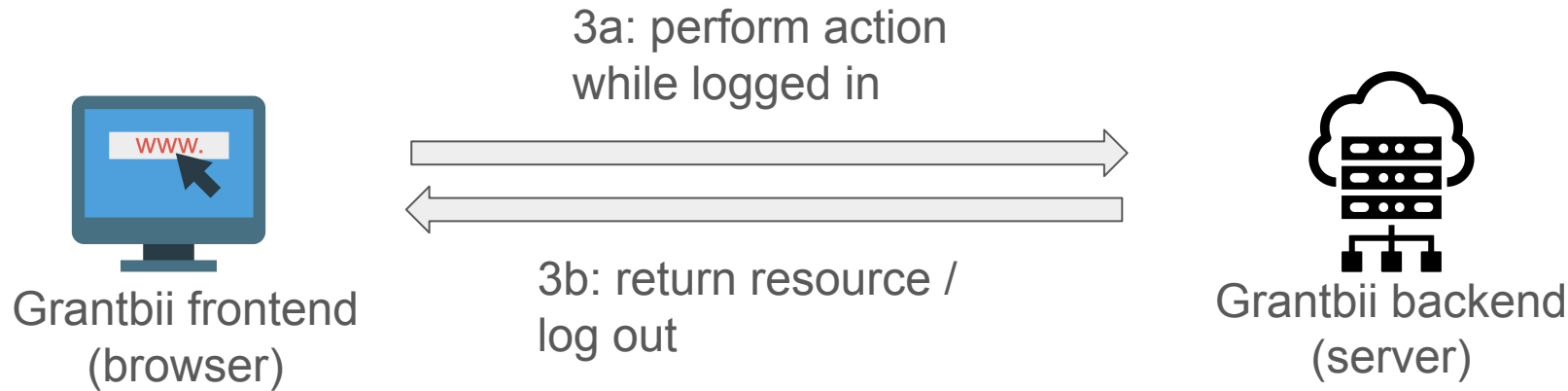
Step 2: Login via magic link

[Skip to behind-the-scenes](#)

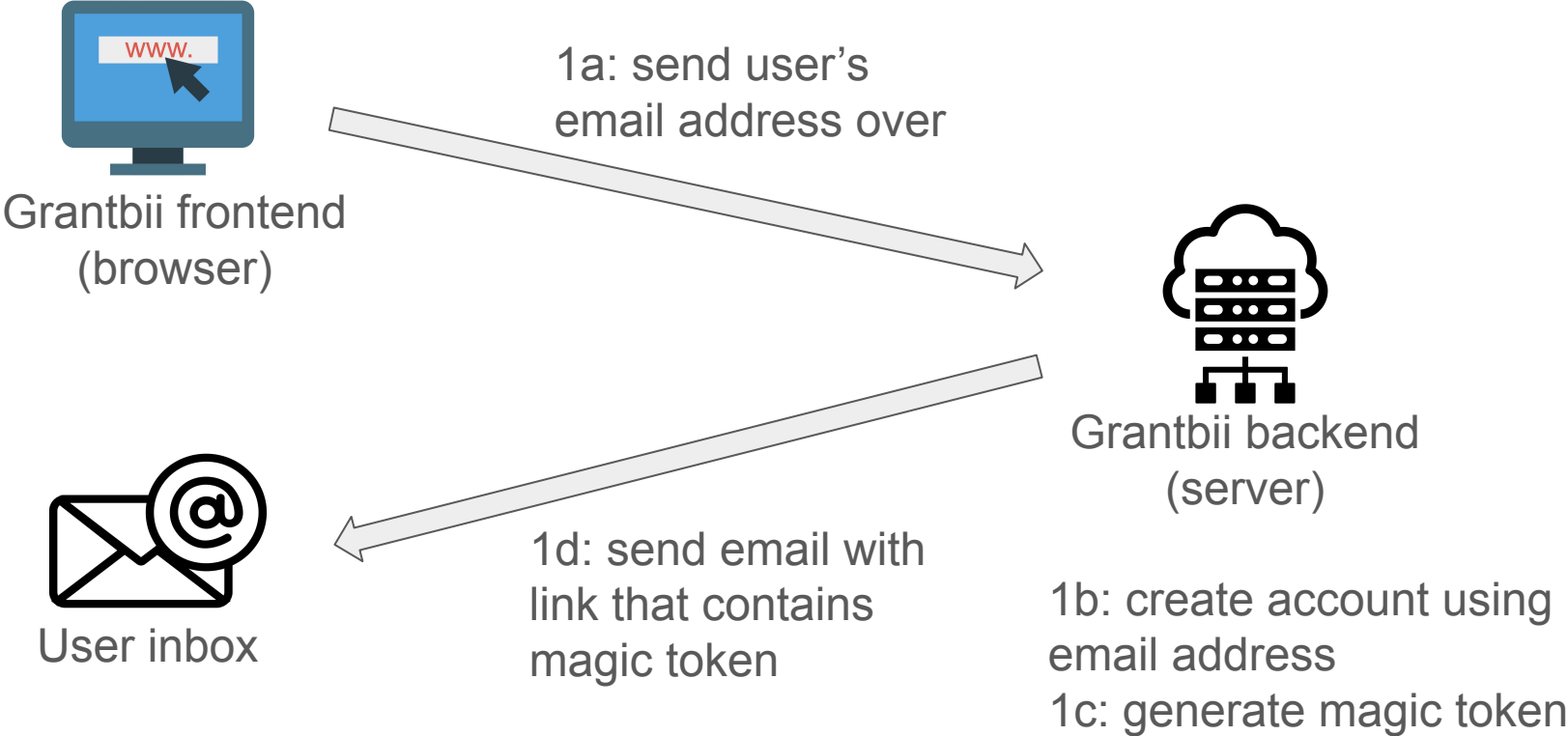


Step 3: Do whatever you need to do

[Skip to behind-the-scenes](#)



Get magic link - behind the scenes



Get magic link - concerns

The email may get intercepted

- The magic token is in plaintext - no protection at all!
- a.k.a. (MITM) Man-in-the-middle attack
- Solution: make the token expire quickly (10 mins)

Login via magic link - behind the scenes



User inbox

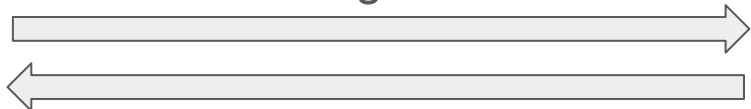


2a: redirect to login page with magic token in URL query params



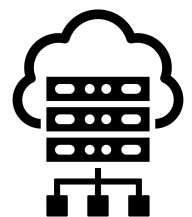
Grantbii frontend (browser)

2b: request for refresh token with valid magic token



2d: store refresh token in cookies

2c: generate refresh token



Grantbii backend (server)

Login via magic link - concerns

User visits a malicious site that runs code (JavaScript) in the browser to scan for tokens

- Including the refresh token stored in cookies
- a.k.a. XSS (Cross Site Scripting)
- Solution: use `httponly=True` flag to make it inaccessible to JavaScript

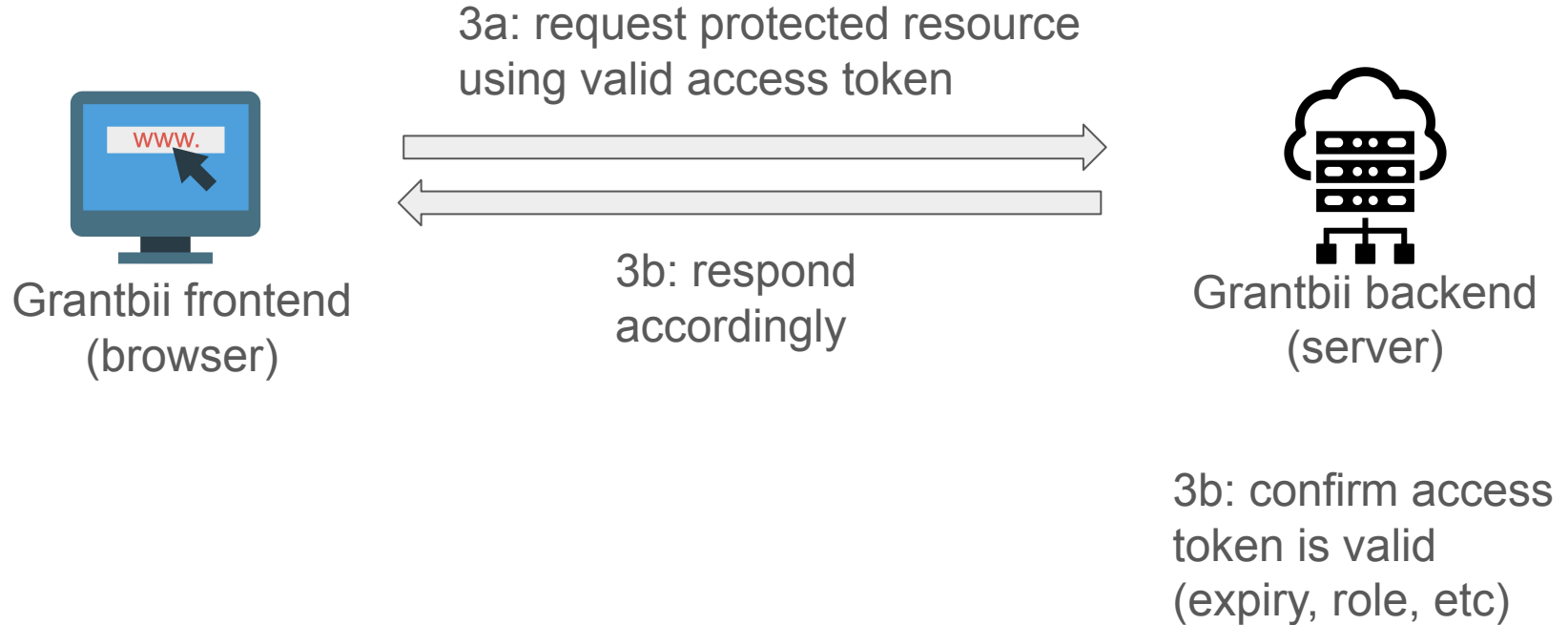
User visits a site that pretends to be Grantbii (e.g. grantbi1.com)

- Site would try to gain access to protected resources by using a legitimate user's credentials
- a.k.a. CSRF (Cross Site Request Forgery)
- Solution: use `samesite=True` flag so that Grantbii backend accepts only refresh token from Grantbii frontend (must be from grantbii.com)

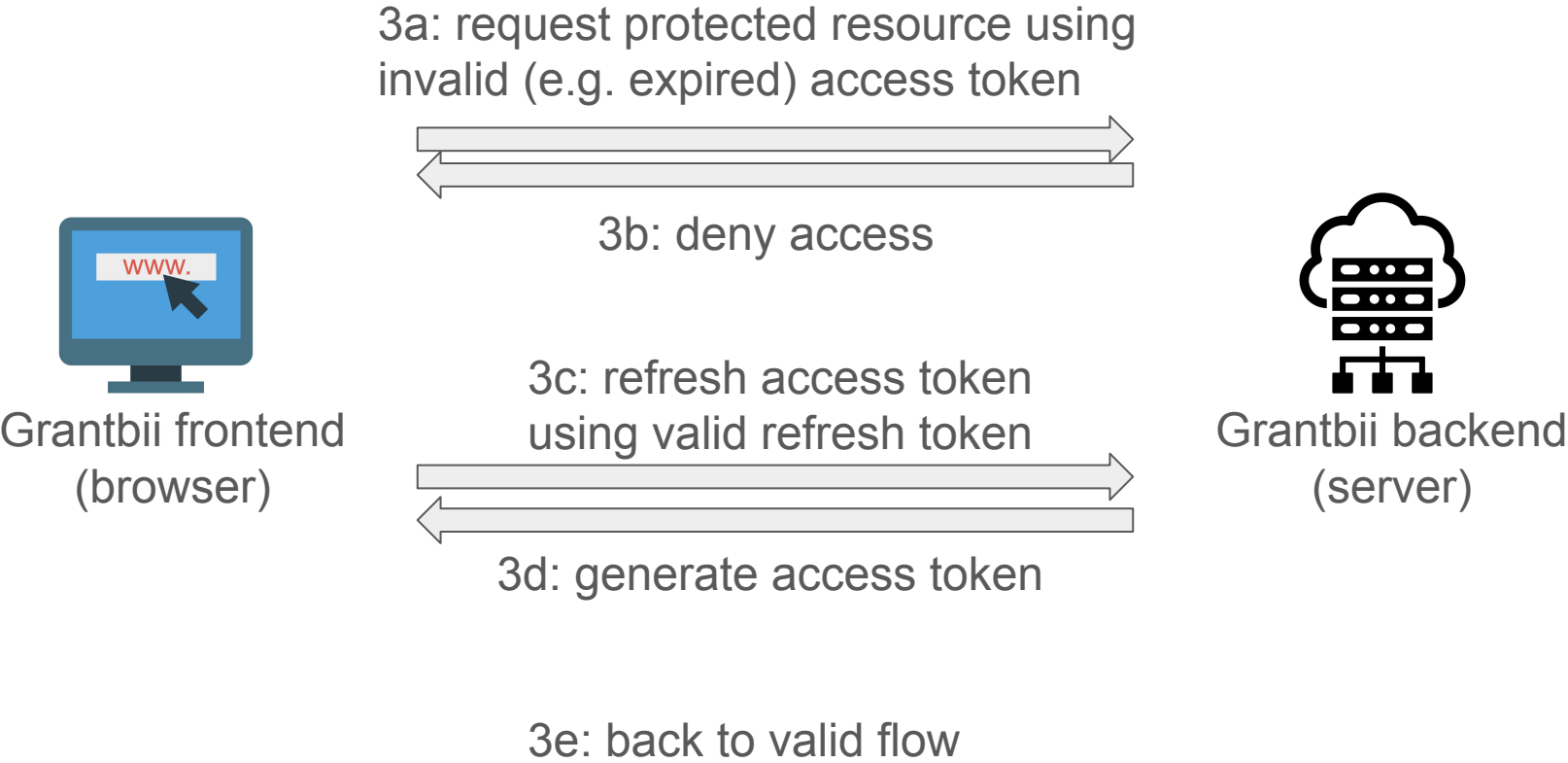
Refresh token may get intercepted while transferring between frontend and backend

- a.k.a. (MITM) Man-in-the-middle attack
- Solution: use `secure=True` flag to restrict transmission mode to HTTPS; encryption means even if the attacker got hold of the token, they can't use it

Do whatever you need to do - behind the scenes (valid token)



Do whatever you need to do - behind the scenes (invalid)



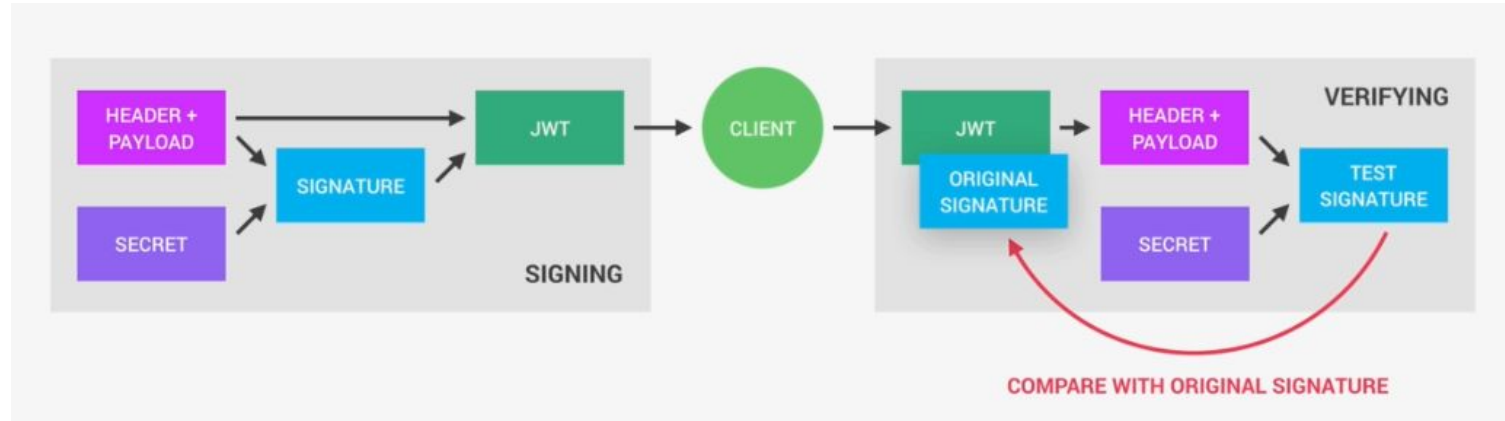
Do whatever you need to do - concerns

Access token is used whenever any protected resources have to be accessed

- i.e. very frequently
- Too many database queries would drastically increase latency
- Solution: encode any auth information (e.g. role) directly in the access token; no need to read from database

Attacker may try forging an access token

- Solution: use JWT (JSON Web Token); [see explanation here](#)



Token Types

Access token

- Used to *access* any protected resources
- Used frequently; any time a protected resource has to be accessed
- Not stored anywhere
- Expires after 15 minutes

Refresh token

- Used to *refresh* the access token
- Used semi-frequently; whenever access token expires
- Stored in cookies on frontend; stored hash in database
- Expires after 30 days

Magic token

- Used for the *magic* link
- Used infrequently; only when refresh token expires
- Stored hash in database
- Expires after 10 minutes

Logout

Just revoke refresh token