SySS THE PENTEST EXPERTS.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF SCIENCE
Communication Networks

Browser Swapping – How to Hack & How to Fix?

By Jonas Primbs, SySS GmbH / University of Tübingen, Germany

https://www.syss.de | https://kn.cs.uni-tuebingen.de

Recap: OAuth 2.0 Authorization Code Grant

Client / Relying Party
`https://client.example.com`

User Agent
user's web browser

Authorization Server / OpenID Provider
`https://as.example.com`

User:

Initiate login session

Remember
`state=xyz`
`nonce=XYZ`

1. Initiate Login

```
GET /login HTTP/1.1
Host: client.example.com
```

2. Authorization Request

```
HTTP/1.1 302 Found
Location: https://as.example.com/authorize?
  response_type=code%20id_token&
  redirect_uri=https%3A%2F%2Fclient.example.com%2Fcallback&
  client_id=s6BhdRkqt3&
  state=xyz&nonce=XYZ&
  code_challenge_method=S256&
  code_challenge=E9Melhoa2OwvFrEMTJguCH…WbuGJSstw-cM
```

3. Authorization Request

```
GET /authorize?
  response_type=code%20id_token&
  redirect_uri=https%3A%2F%2Fclient.example.com%2Fcallback&
  client_id=s6BhdRkqt3&
  state=xyz&nonce=XYZ&
  code_challenge_method=S256&
  code_challenge=E9Melhoa2OwvFrEMTJguCH…WbuGJSstw-cM HTTP/1.1
Host: as.example.com
```

4. **User** AuthN & AuthZ

5. Authorization Response

```
HTTP/1.1 302 Found
Location: https://client.example.com/callback?
  state=xyz&
  code=SplxlOBeZQQYbYS6WxSbIA&
  id_token=ey…XYZ…
```

6. Authorization Response

```
GET /callback?
  state=xyz&
  code=SplxlOBeZQQYbYS6WxSbIA&
  id_token=ey…XYZ… HTTP/1.1
Host: client.example.com
```

Verify
`state:xyz=xyz`
`nonce:XYZ=XYZ`

7. Token Request

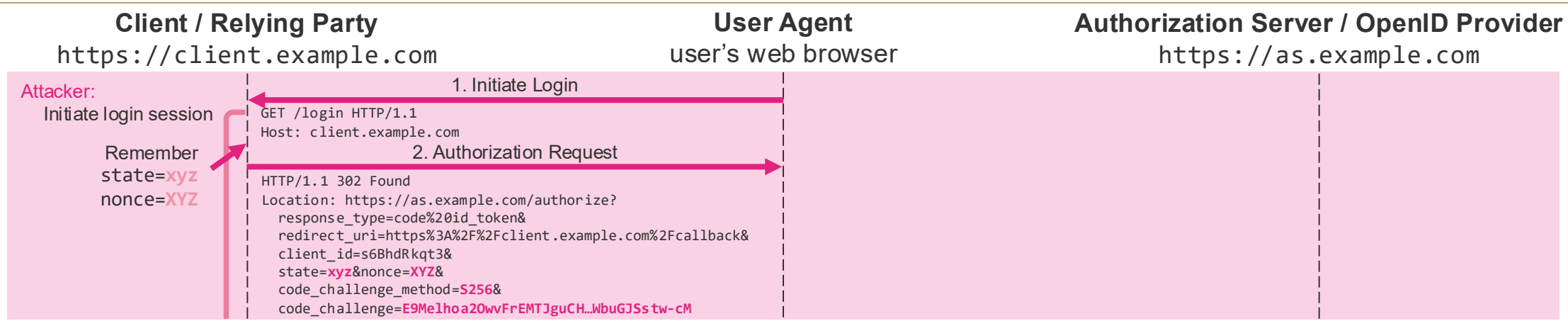
```
POST /token HTTP/1.1
Host: as.example.com

grant_type=authorization_code&
code=SplxlOBeZQQYbYS6WxSbIA&
client_id=s6BhdRkqt3&
redirect_uri=https%3A%2F%2Fclient.example.com%2Fcallback&
code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

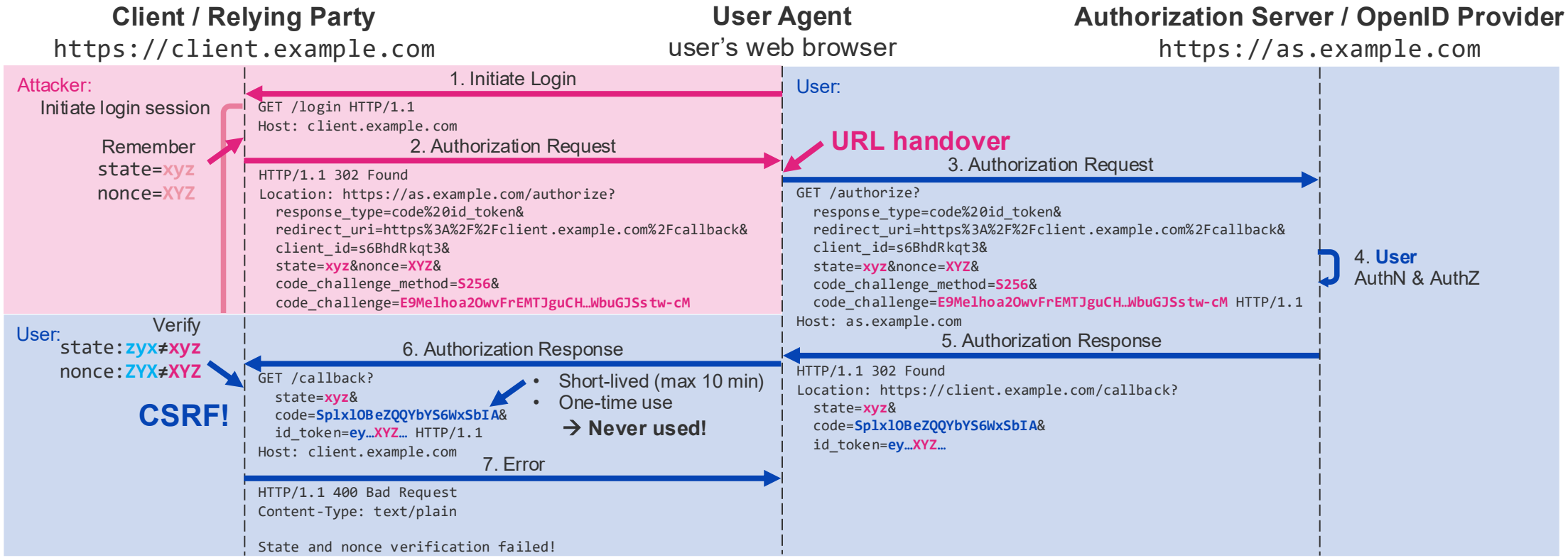
Finish login session as **User**

Demo Time

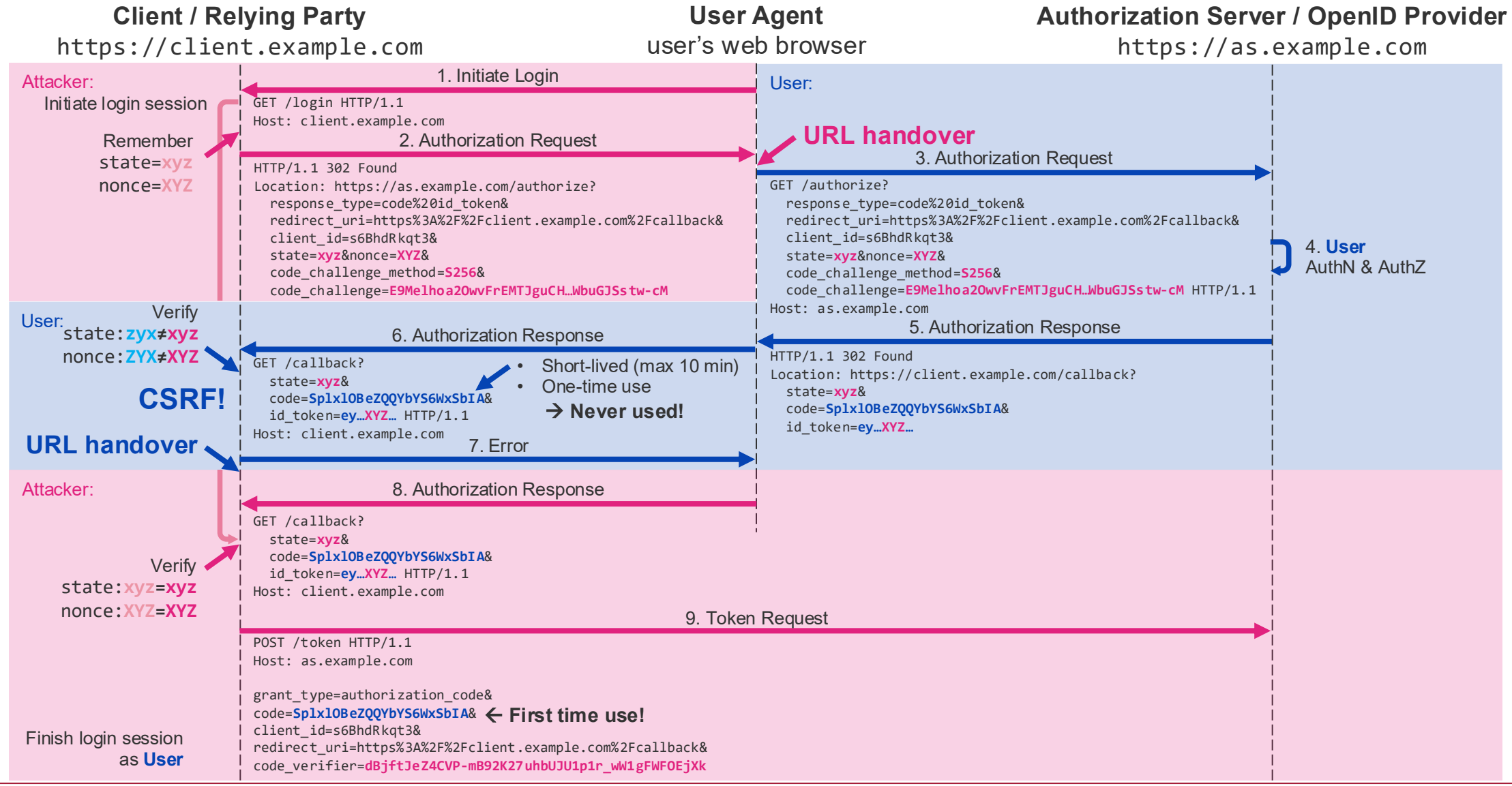
Browser Swapping Attack – Step 1 (Attacker)



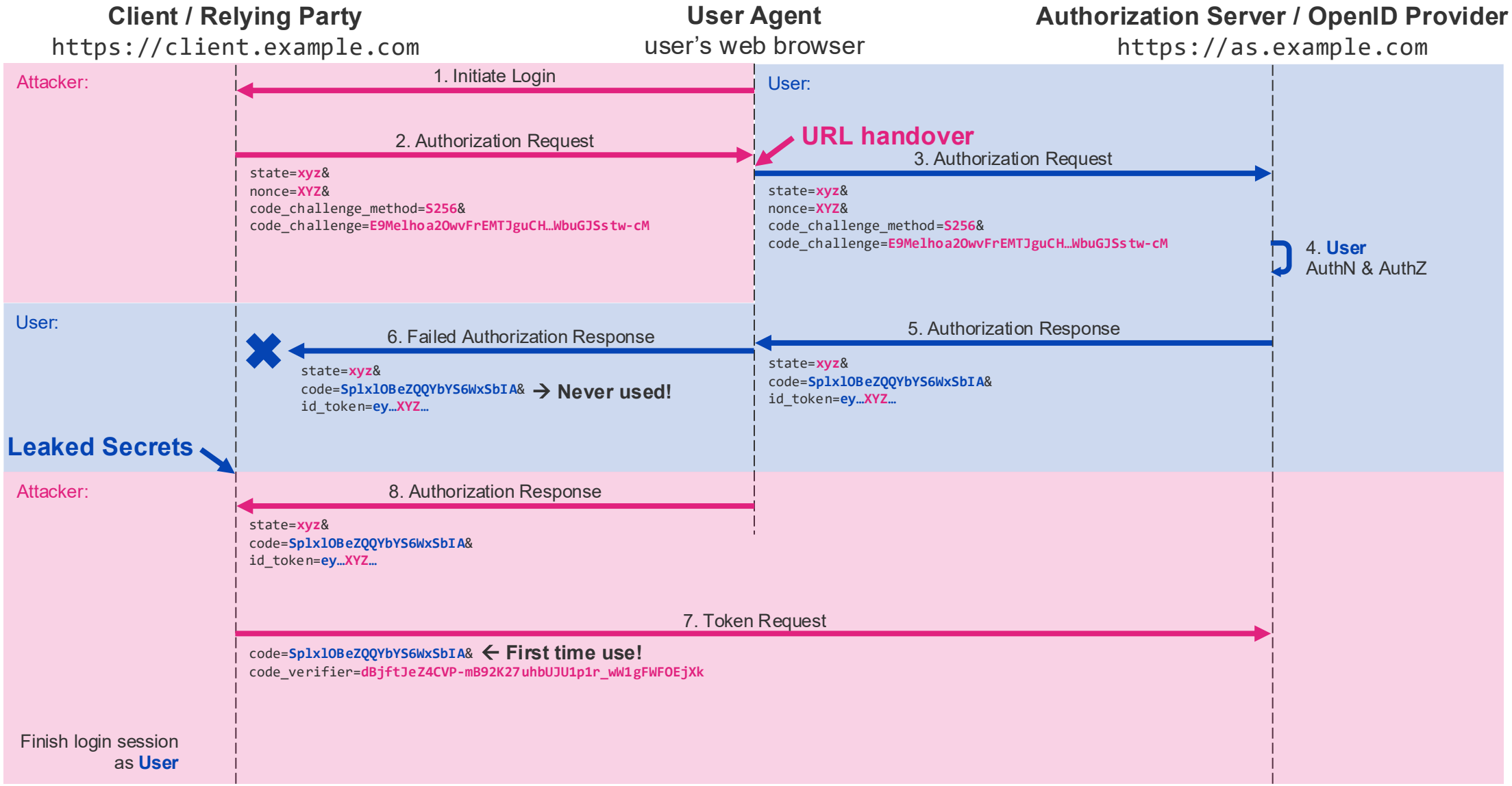
Browser Swapping Attack – Step 2 (User)



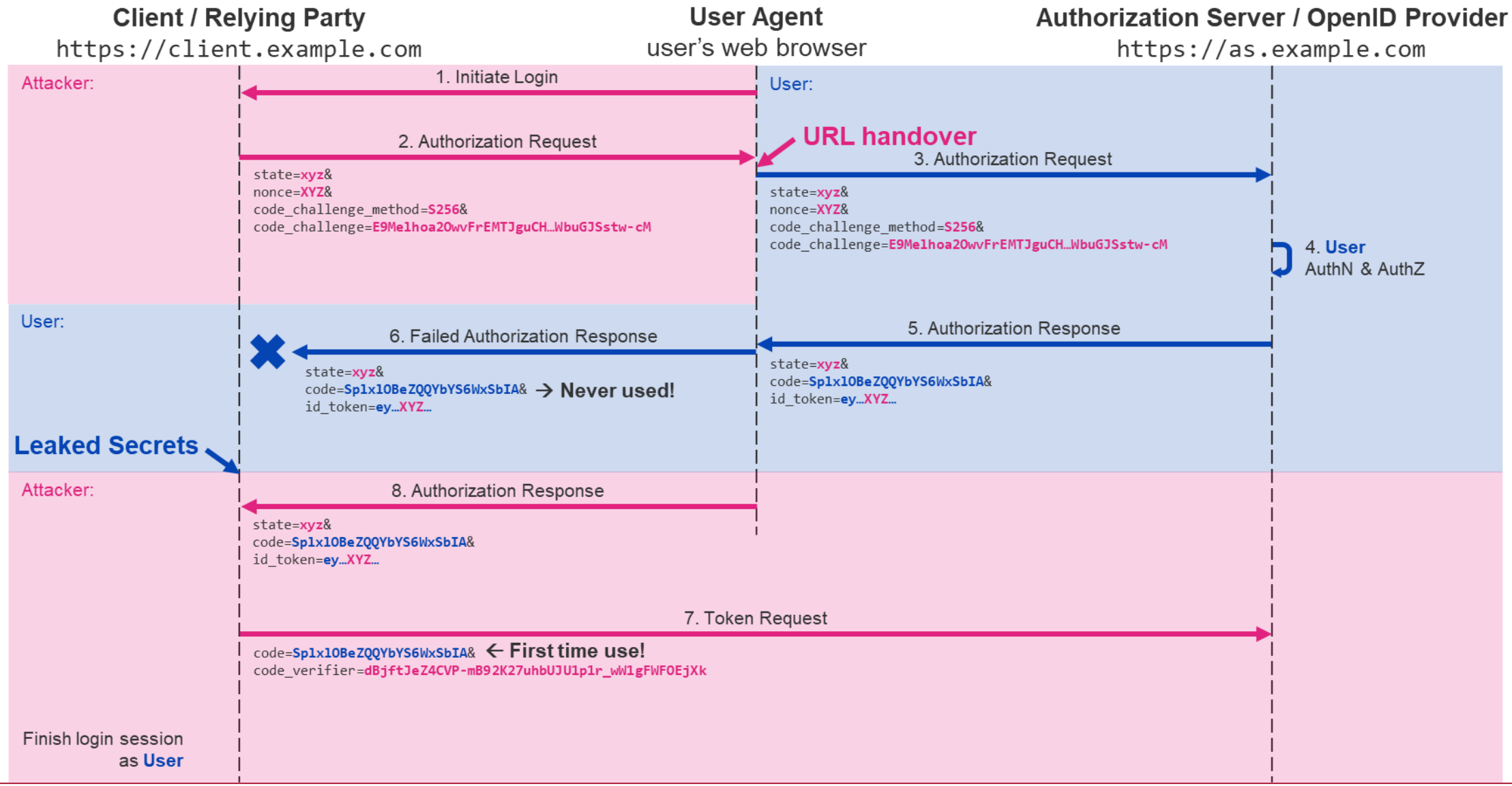
Browser Swapping Attack – Step 3 (Attacker)



Browser Swapping Attack – Abstract Concept

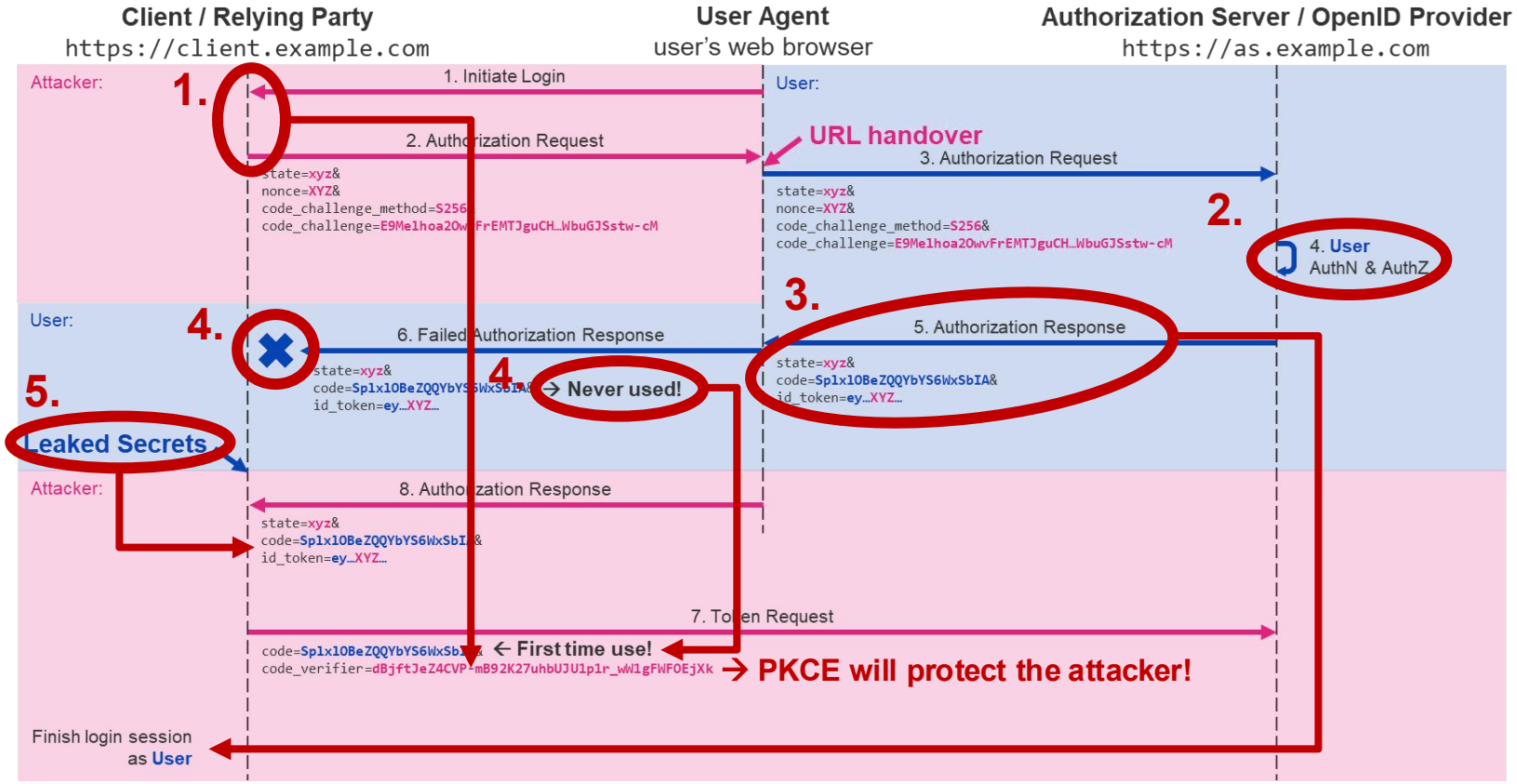


Browser Swapping Attack – Abstract Concept

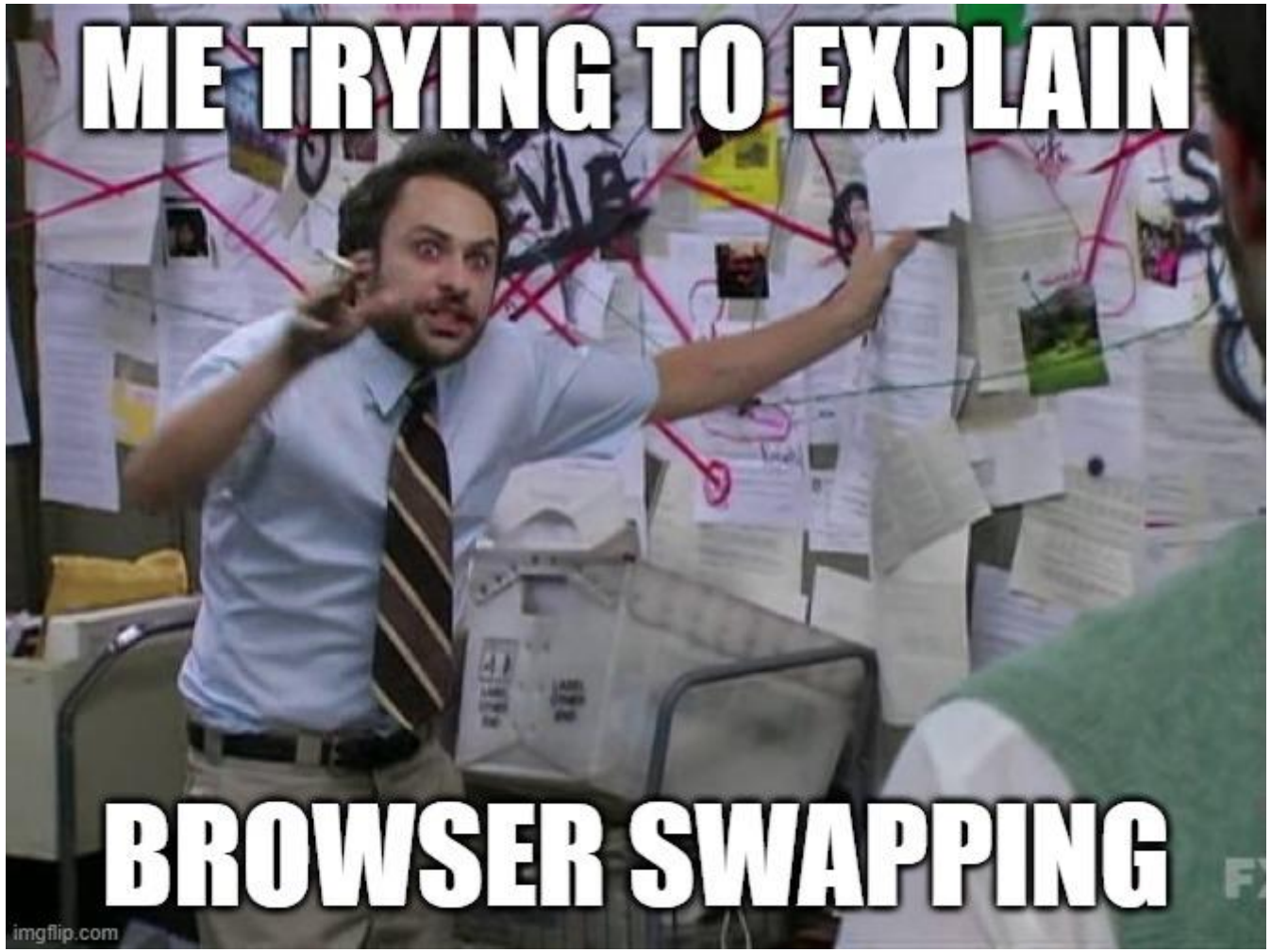


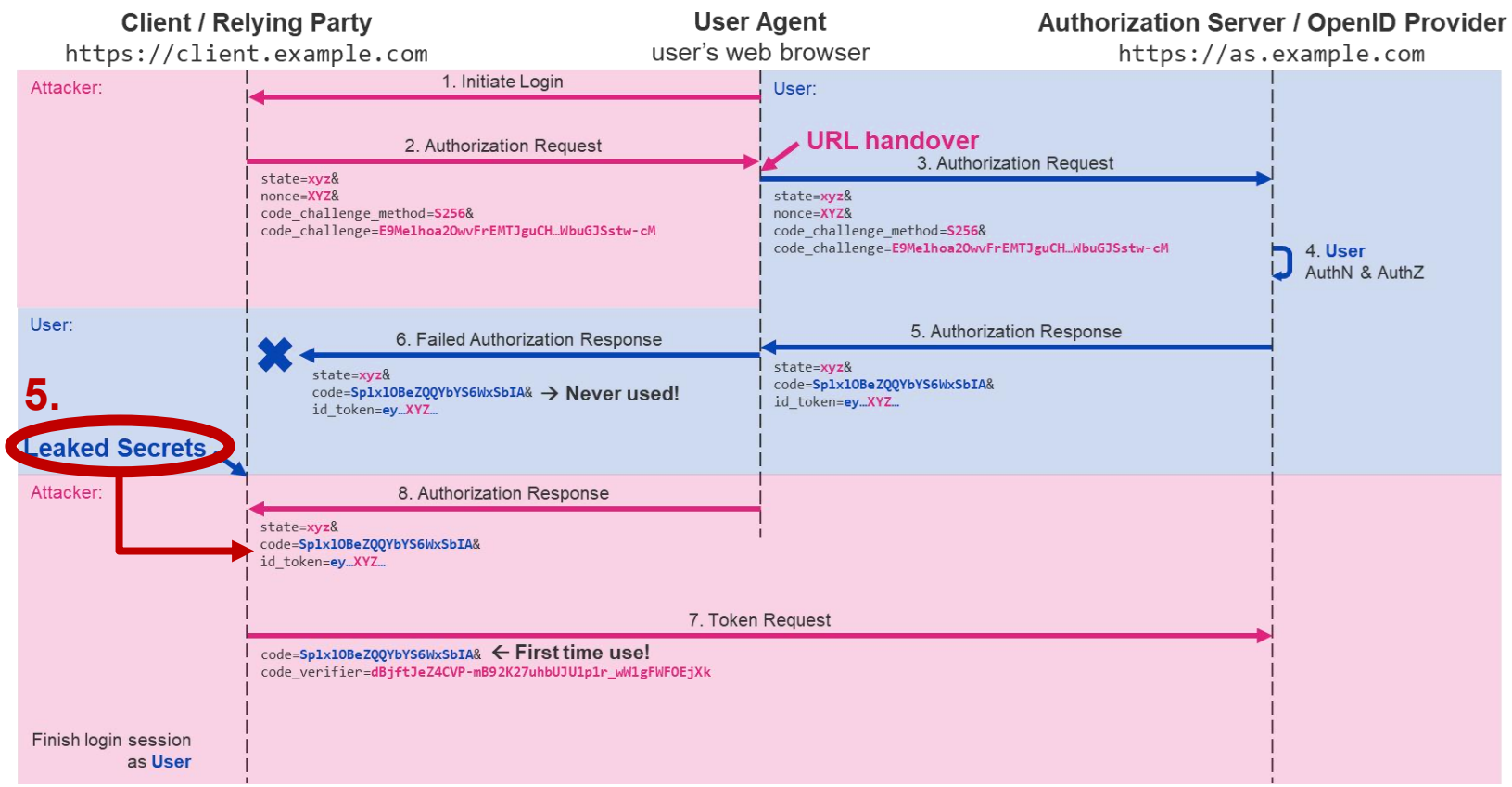
Analysis: Why is Browser Swapping possible?

1. The attacker initiates the authorization flow
2. The user grants the authorization request
3. The AS/OP issues the user's authorization code with the attacker's state and nonce
4. The user's authorization code (one-time use) is not used by the user's Client/RP
5. The user leaks their authorization code to the attacker



Questions so far?



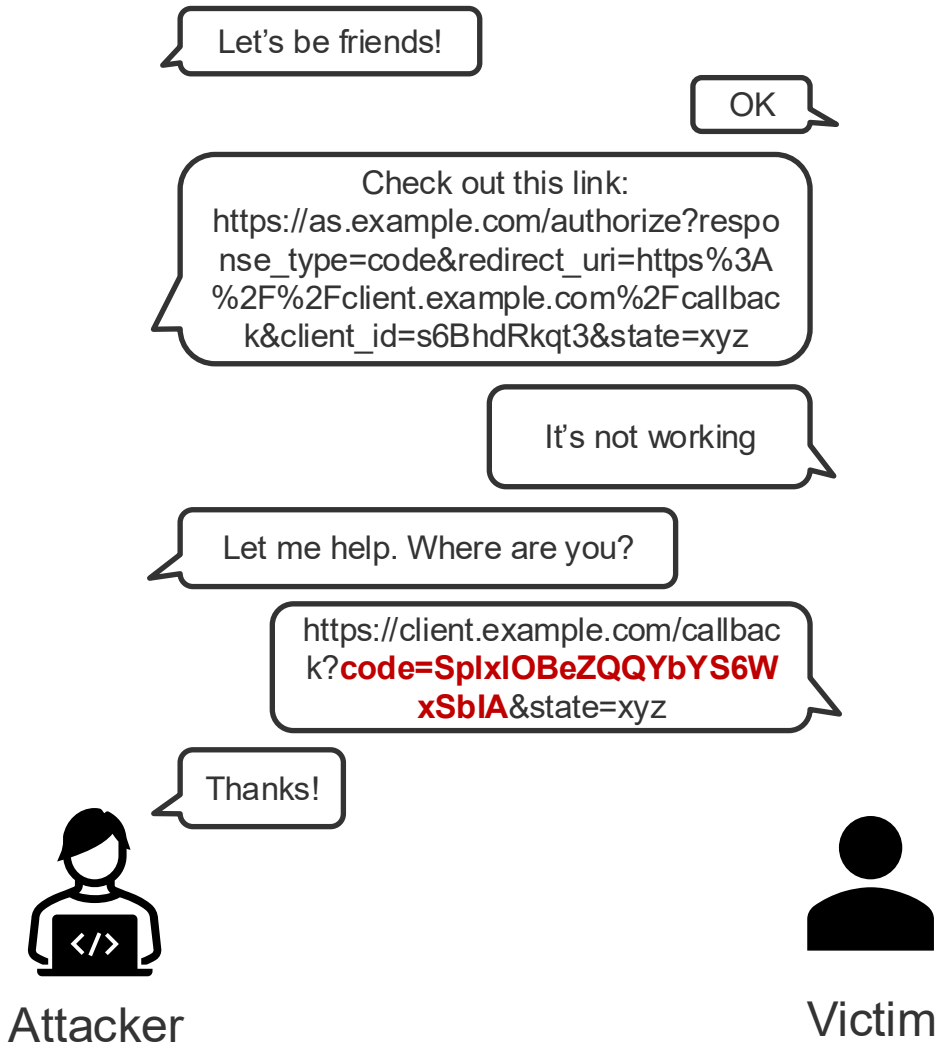


5. Leaked Secrets

Leaked Secrets #1: URL Sharing

Option 1: Social Engineering with Trust Relationship

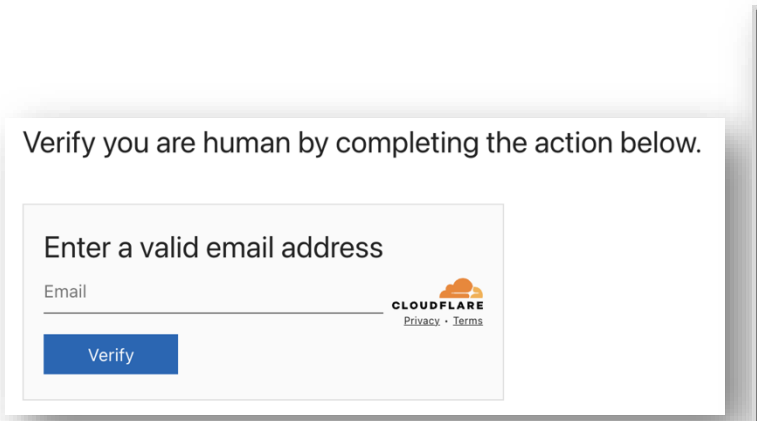
- Attack Scenario
  1. Attacker establishes a trust relationship with the victim
  2. Attacker sends authorization request URL to victim
  3. Victim logs in; callback fails
  4. Attacker pretends to help and asks for callback URL
- Does not scale well
  - Requires much time to establish trust relationship
    - Chat bots can do this for the attacker

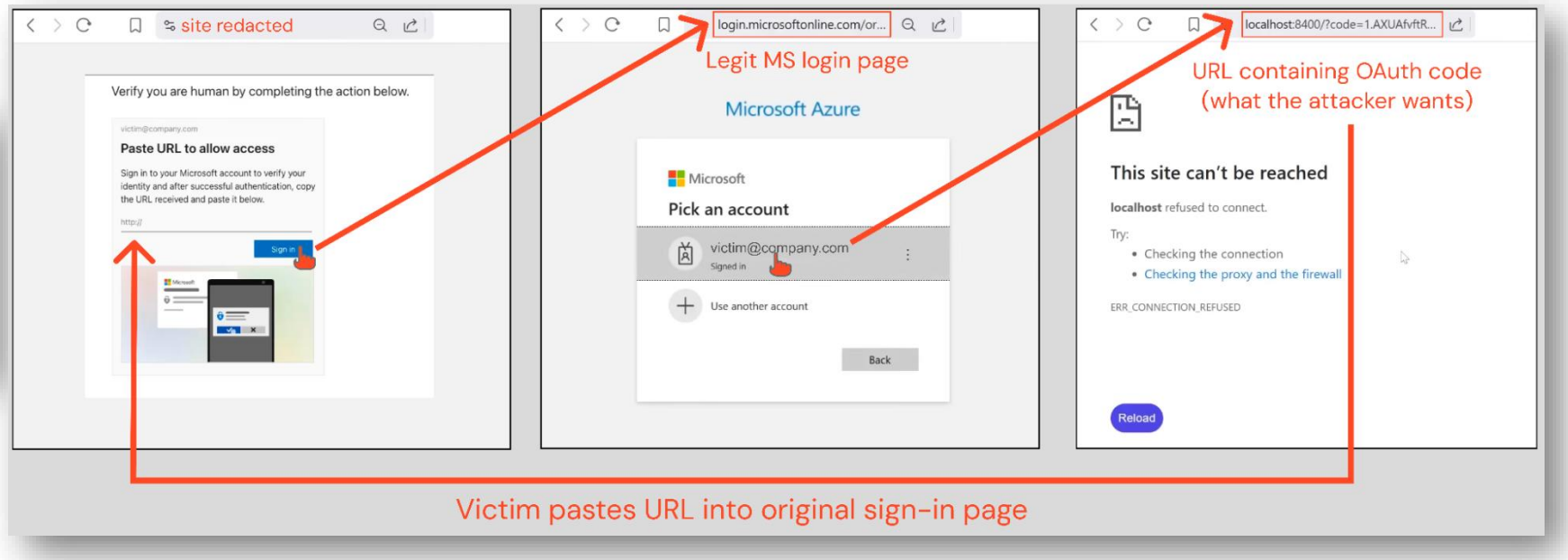


Leaked Secrets #1: URL Sharing

Option 2: Legitimate Reasons

- Actively misused to attack Azure CLI: see ConsentFix
  1. Attacker pretends to be Cloudflare's bot detection and asks for email address
  2. Attacker initiates flow to authorize the Azure CLI (`redirect_uri=localhost`)
  3. Callback URI not found on the user's device → user pastes callback URI to "bot detector" form





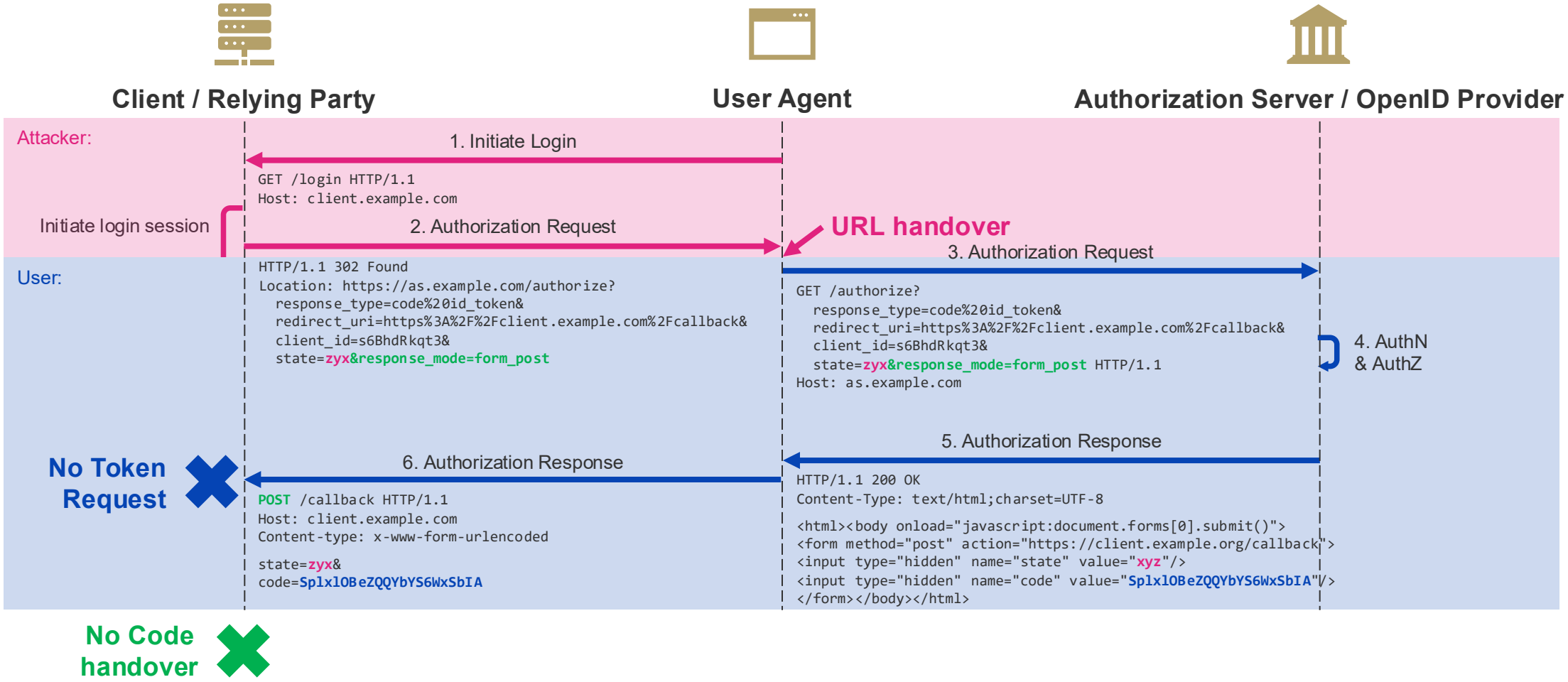
Source: https://pushsecurity.com/blog/consentfix

Solutions?


PREVENT USERS FROM SHARING SECRETS
THAT WOULD BE GREAT
imgflip.com

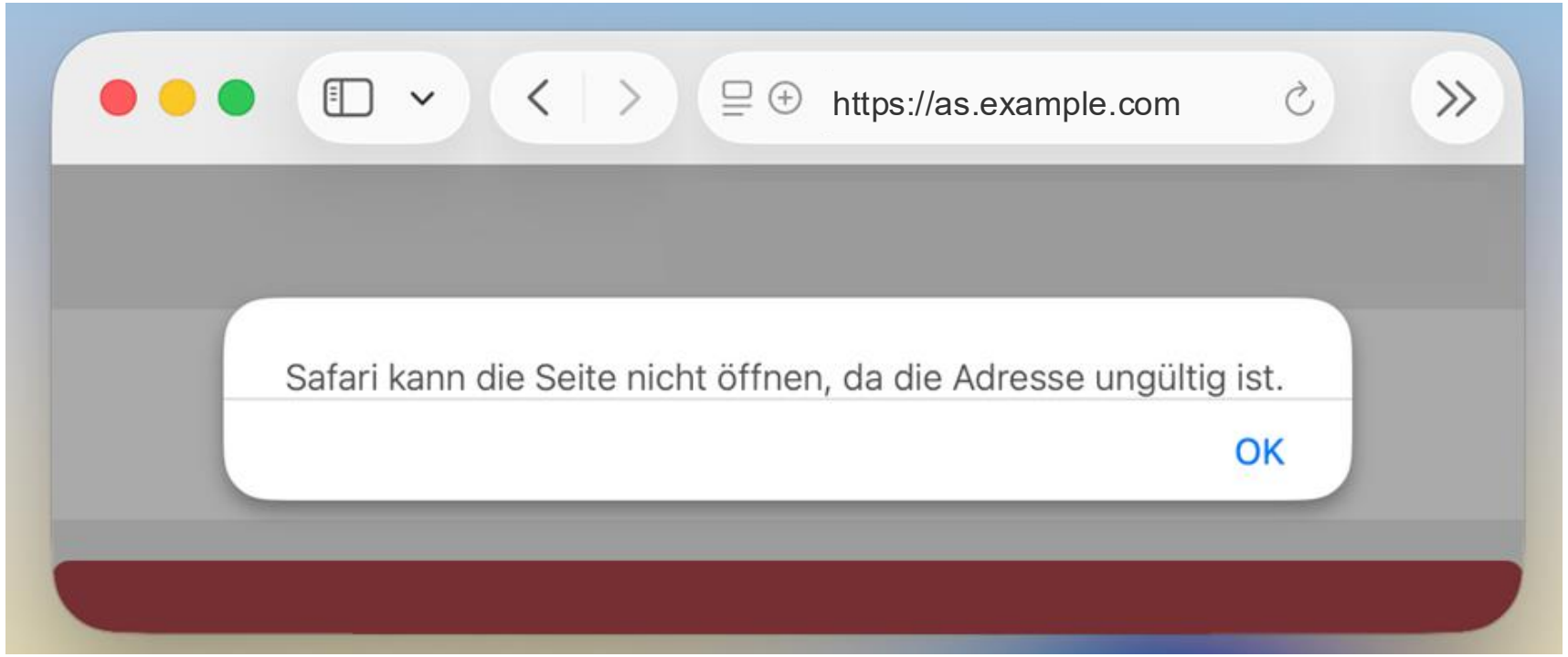
Solution 1: Response Mode Form POST

- Enforcing **`response_mode=form_post`** (OpenID standard) is an **aftercare** mechanism making authorization codes not sharable in a URL



Solution 2: App Links

- App Links are **affected in theory**
  - Example redirect URI:
    `com.example.client:///callback?state=zyx&code=SplxlOBeZQQYbYS6WxSbIA&id_token=ey…ZYX…`
- However, they are **not affected in practice** because App Link URIs cannot be copied or shared

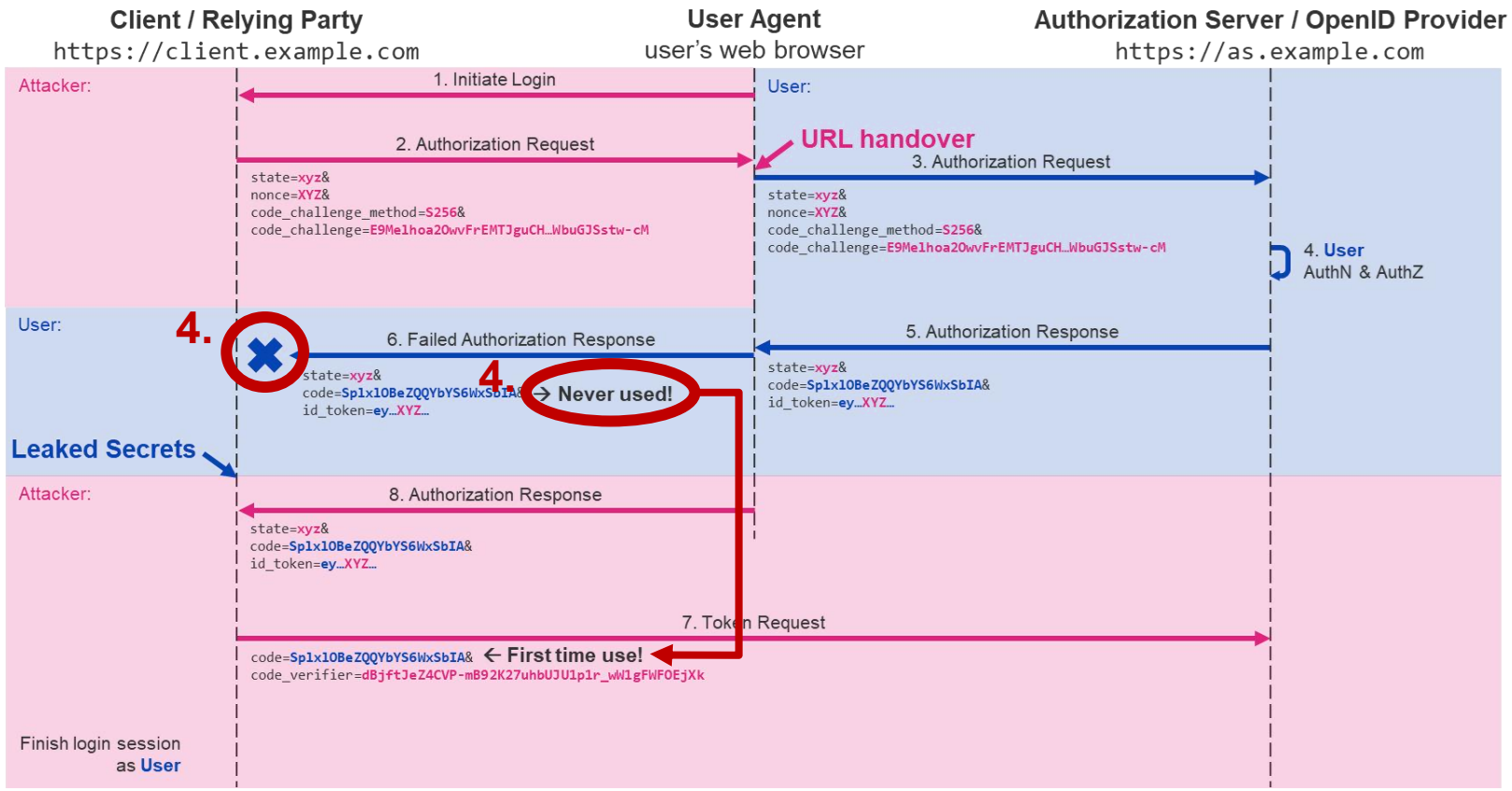


Solution 3: Short-lived Authorization Code

- Short lifetime reduces risk of an authorization code being shared with the attacker in time

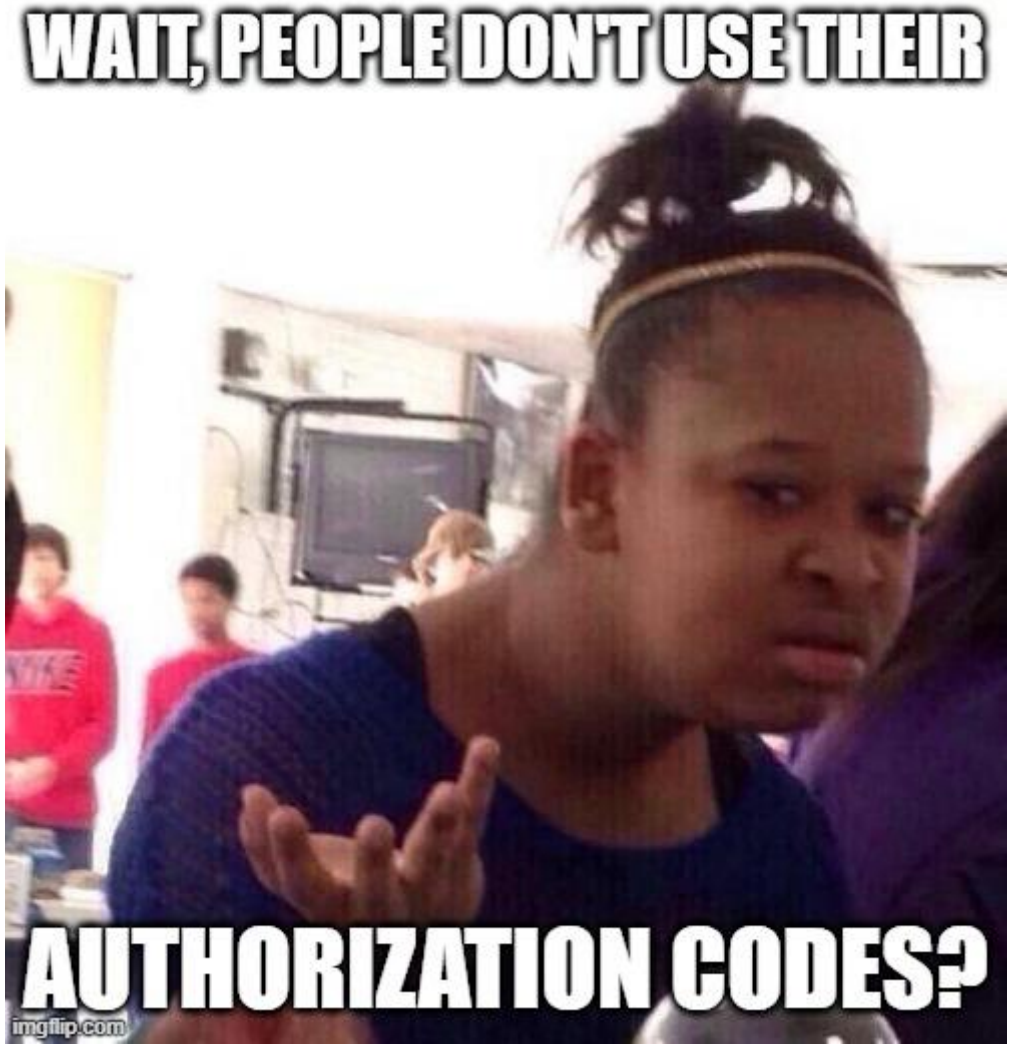
- **Problems**
  - Users can be trained to go faster
  - Login flows on slow clients/RPs + slow Internet connections may break

- **Suggestion**: 5 seconds validity period



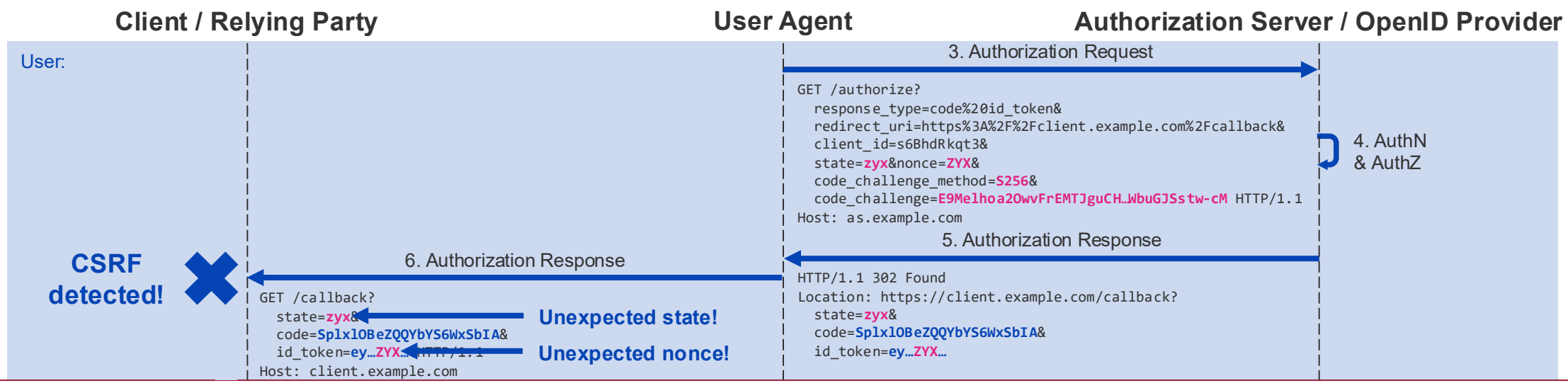
4. Unused One-Time Secrets (OTS)

Unused OTS


WAIT, PEOPLE DON'T USE THEIR
AUTHORIZATION CODES?
imgflip.com

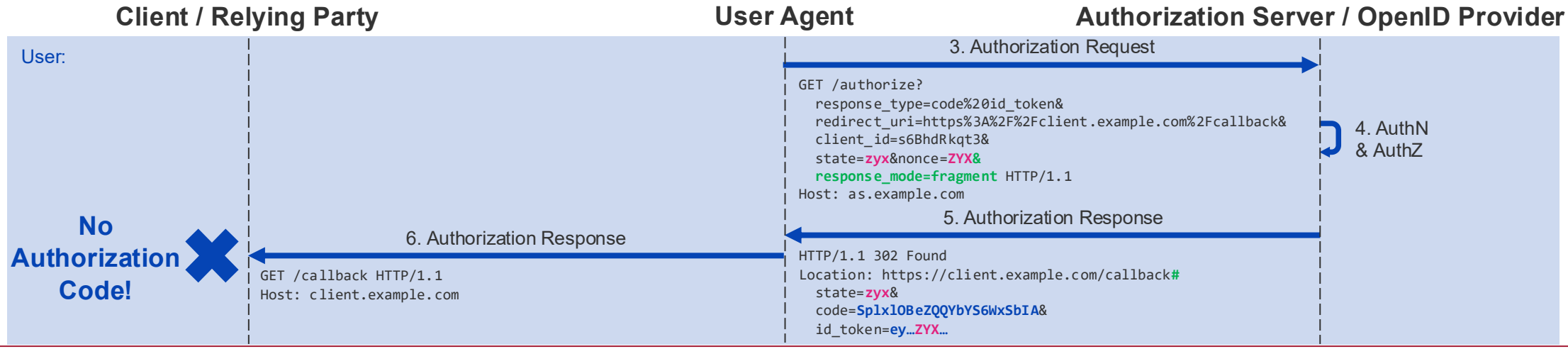
Unused OTS #1: CSRF Protection

- **Issue**: RFC 6749 forces the client to ignore the authorization code without a matching state
- **Result**: Authorization code remains unused in redirect URI
- **Solutions**
  - Client redirects to error URI, e.g., `https://client.example.com/error?error=invalid_state`
    → standardized in RFC 6749, but needs clarification



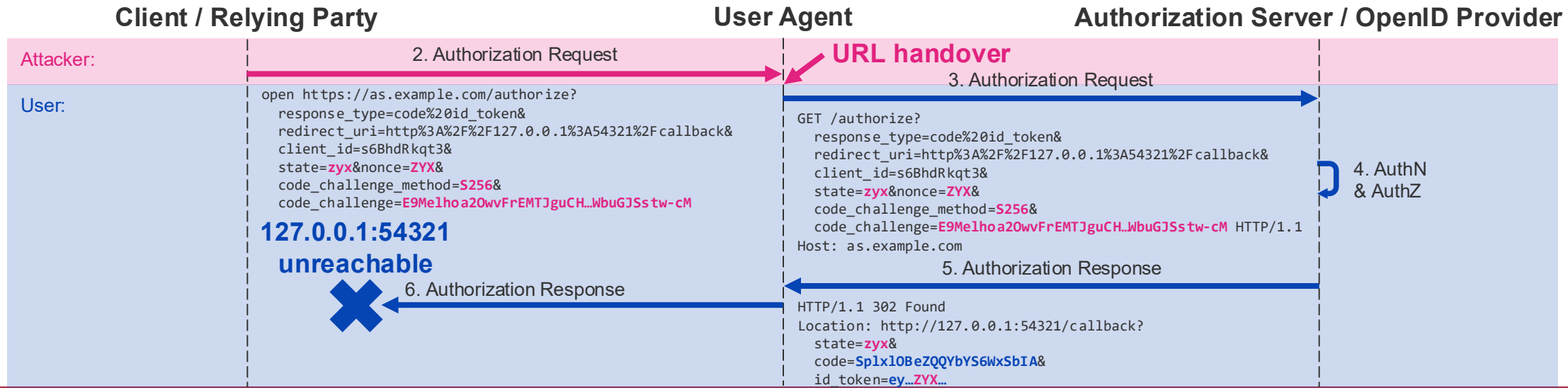
Unused OTS #2: Response Mode Mutation

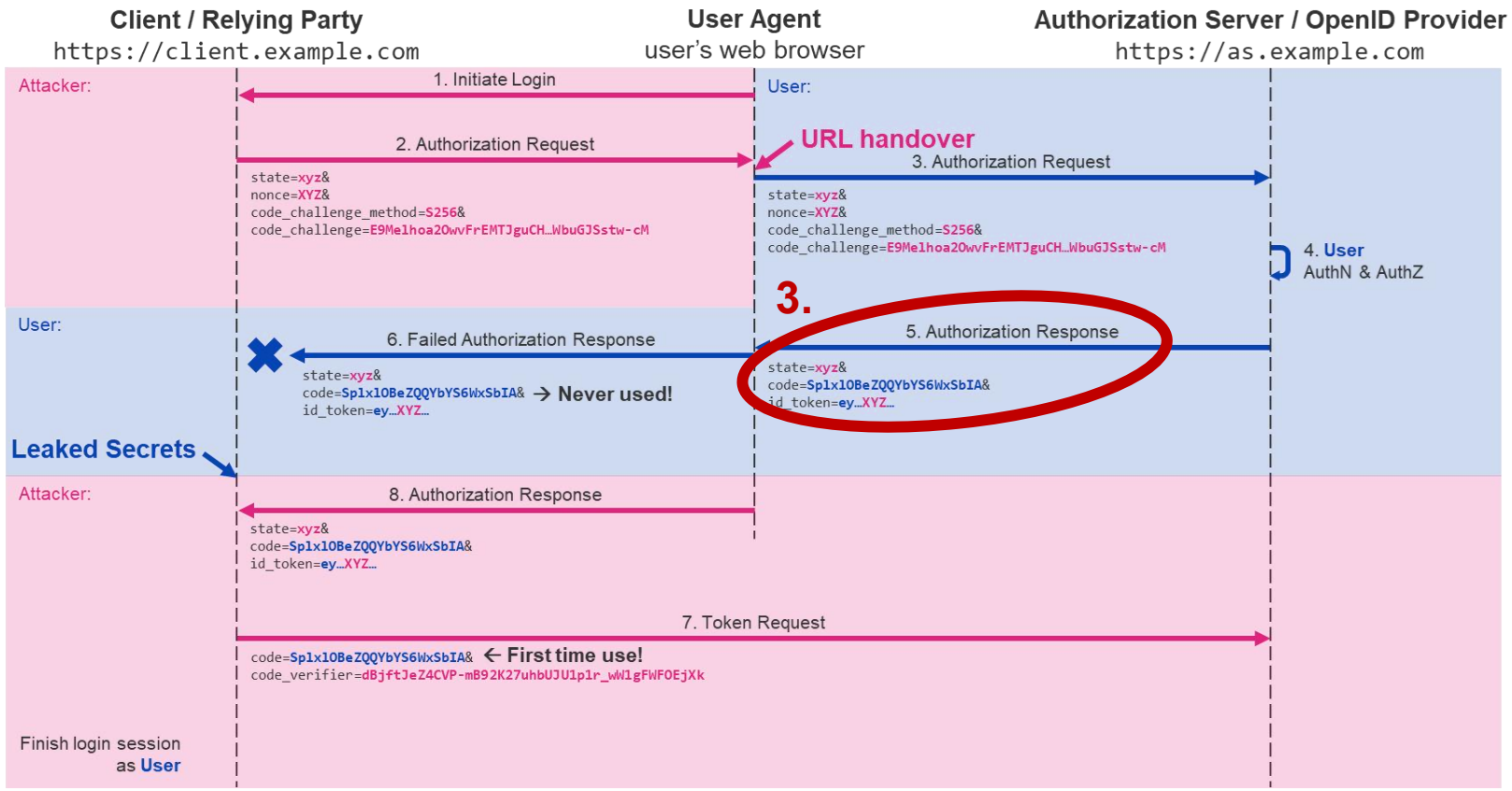
- **Issue**: Attackers change `response_mode` parameter to a response mode that the client does not expect
  - Example: `response_mode=query` to `response_mode=fragment`
- **Result**: Client does not find authorization code in query parameters; Browsers retain fragment part in URL even after redirect to the error URI
  - Example: `https://client.example.com/error?error=missing_code#state=zyx&code=SplxlOBeZQQYbYS6WxSbIA`
- **Solutions → Both not covered by specs!**
  - **Enforce expected response mode** at the authorization server
    - Pushed Authorization Requests (PAR, RFC 9126) for confidential Backend Web Apps work as well
  - **Clear fragment** part when redirecting to error URI with a trailing #
- **Problem**: Client/RP must be reachable!



Unused OTS #3: Unreachable Client

- **Issue**: Loopback interface of attacker's client is not reachable on victim's device
  - Actively misused: see ConsentFix
  - Same issue for HTTPS domains with network layer attacks (e.g., ARP Spoofing), blocked URLs (e.g., state with SQL injection string may be blocked by WAF), or not processable URLs (long state parameter)
  - If mobile app with HTTPS domain handler is not installed: fallback to previous point
- **Result:** Browser shows unreachable error; authorization code remains shareable in URL bar
- **Solution**: `response_mode=form_post` makes authorization code not sharable

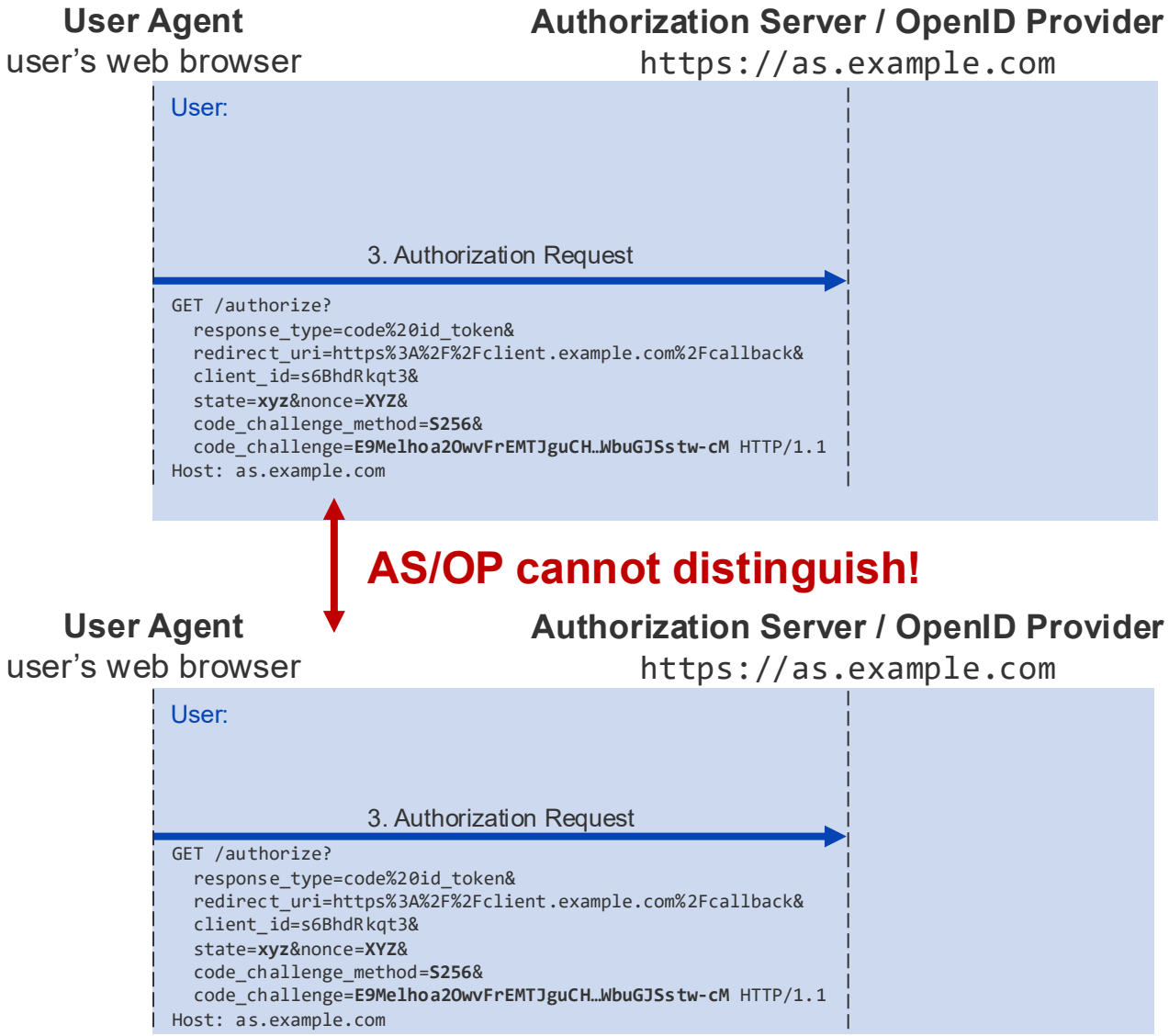




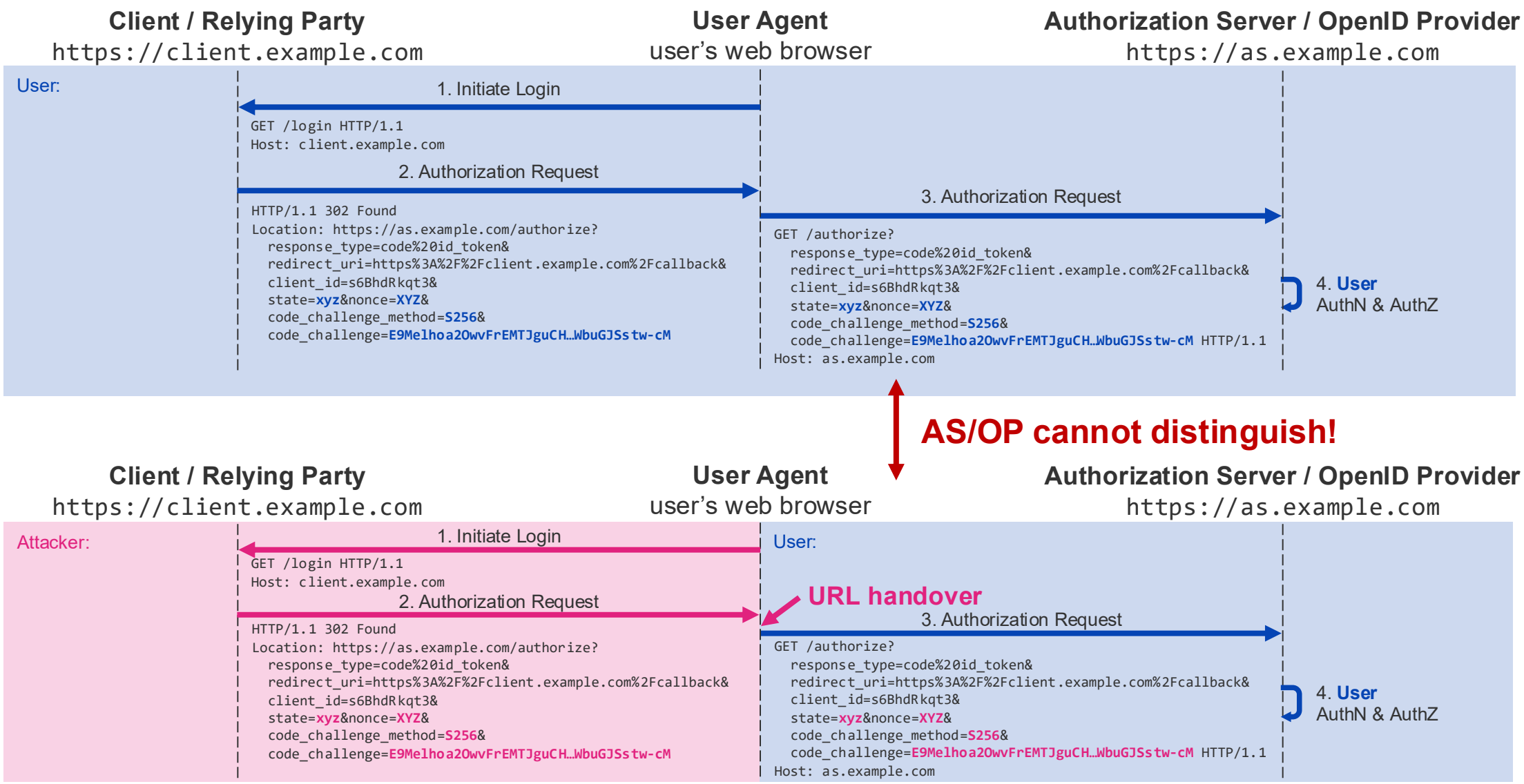
3. Secret Issuance

Secret Issuance

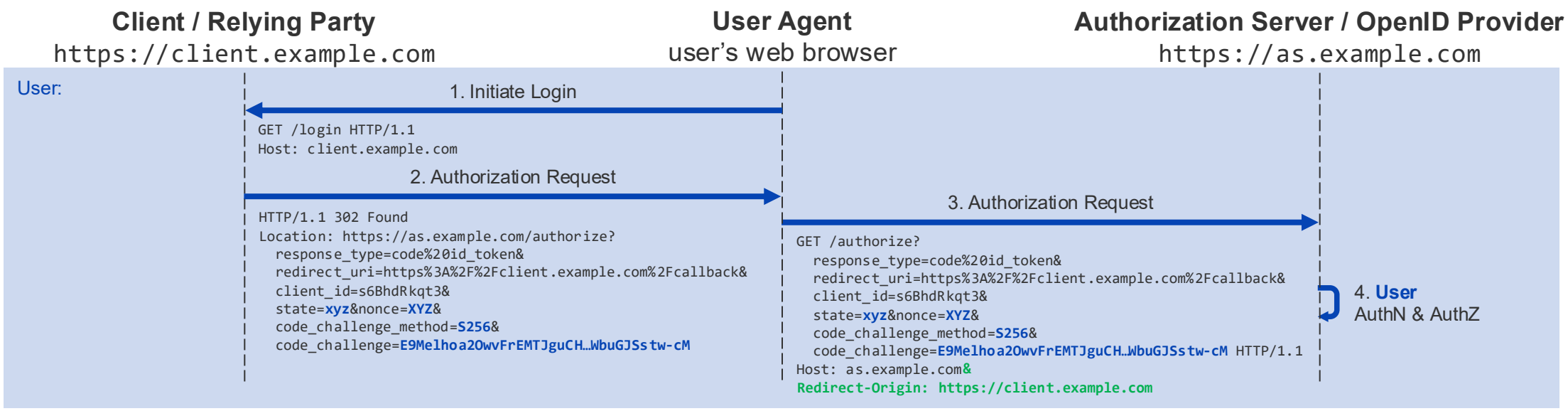
- **Problem**: The AS/OP cannot distinguish a legitimate request initiated by the user from a malicious request injected by the attacker to the user

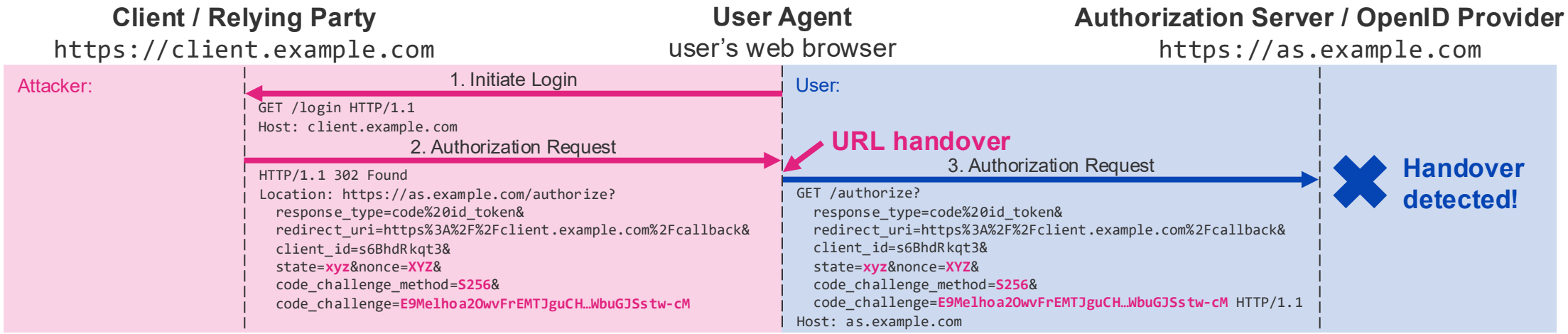


Secret Issuance



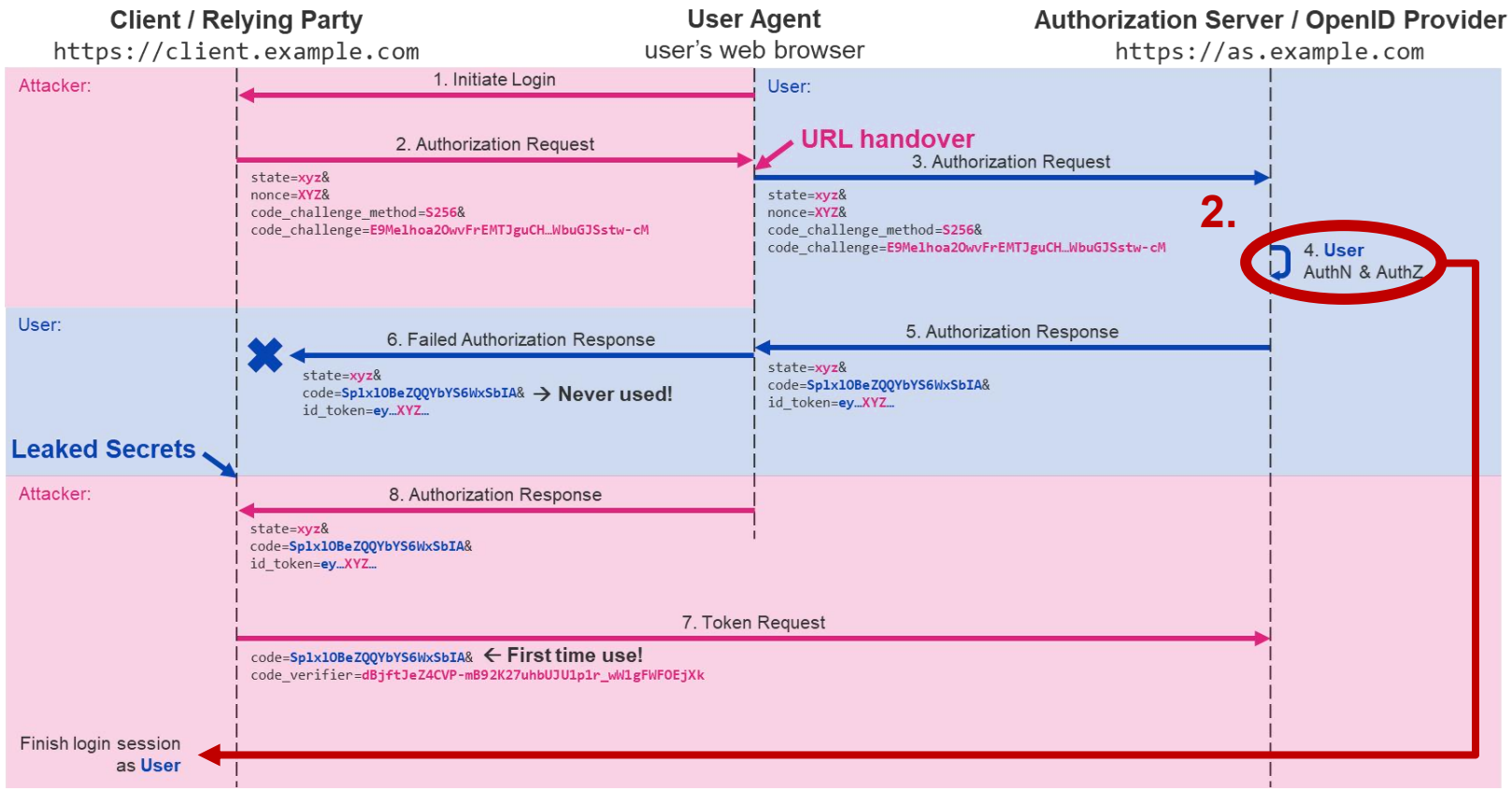
Solution: HTTP Redirect Headers





HTTP Redirect Headers

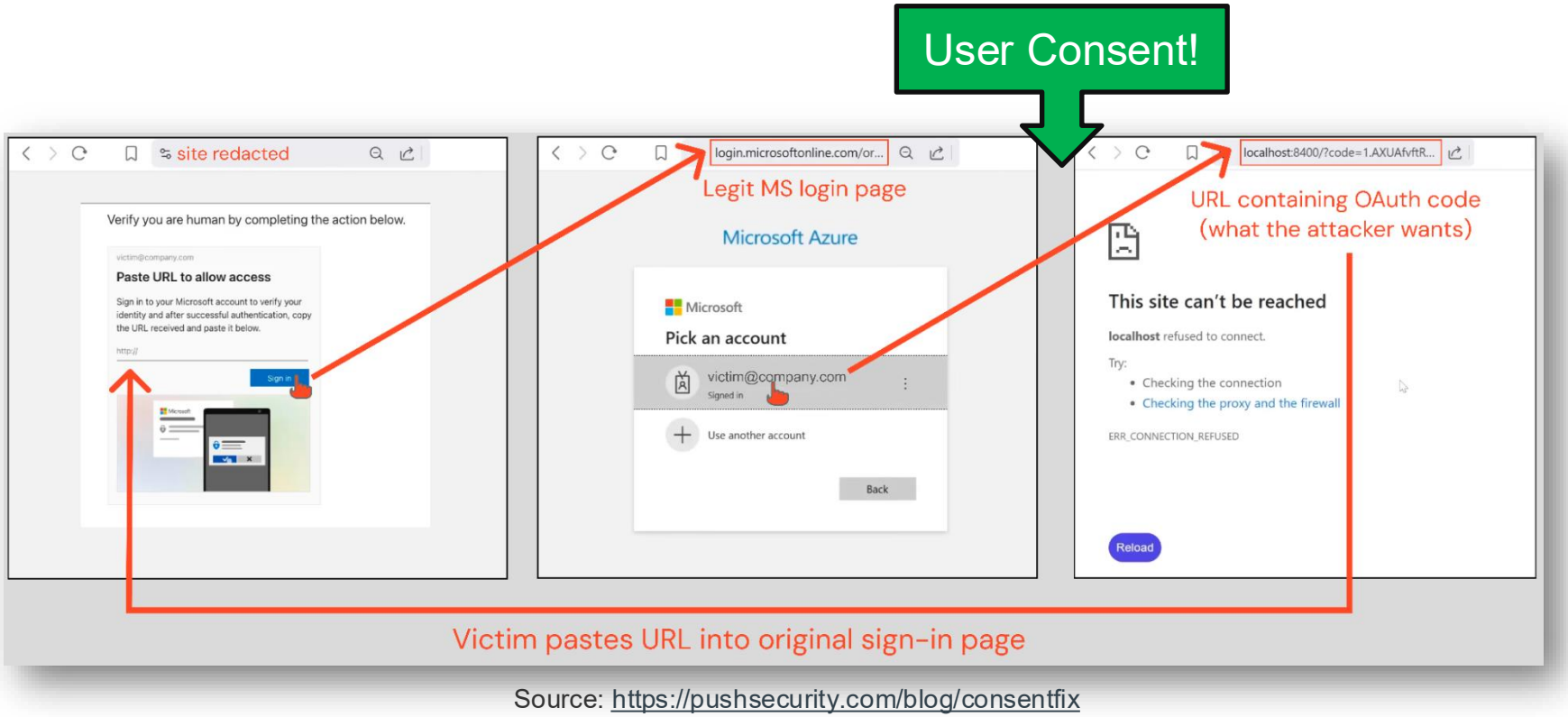
- **Problems**
  - Not yet applicable in today's browsers
    - Requires finished standardization for web browsers
    - Requires implementation to all web browsers
    - Requires users to update their web browser
  - Not yet applicable to native apps with native browser login
    - Requires all before, **plus**:
    - Requires standardization for operating system APIs
    - Requires API implementation to all operating systems
    - Requires users to update their operating system
    - Requires implementation of OS-API to all web browsers
    - Requires implementation of standards to native apps
    - Requires users to update their native app
  - Privacy concerns
    - Every redirect target knows where the user has been redirected from (not only the AS/OP!)



2. User Consent

User Consent

- **Specs**: "[…] the authorization server authenticates the resource owner **and obtains an authorization decision** (by asking the resource owner or by establishing approval via other means)." - RFC 6749
- **Reality**: Client requests authorization → AS authenticates RO → AS issues authorization code / tokens

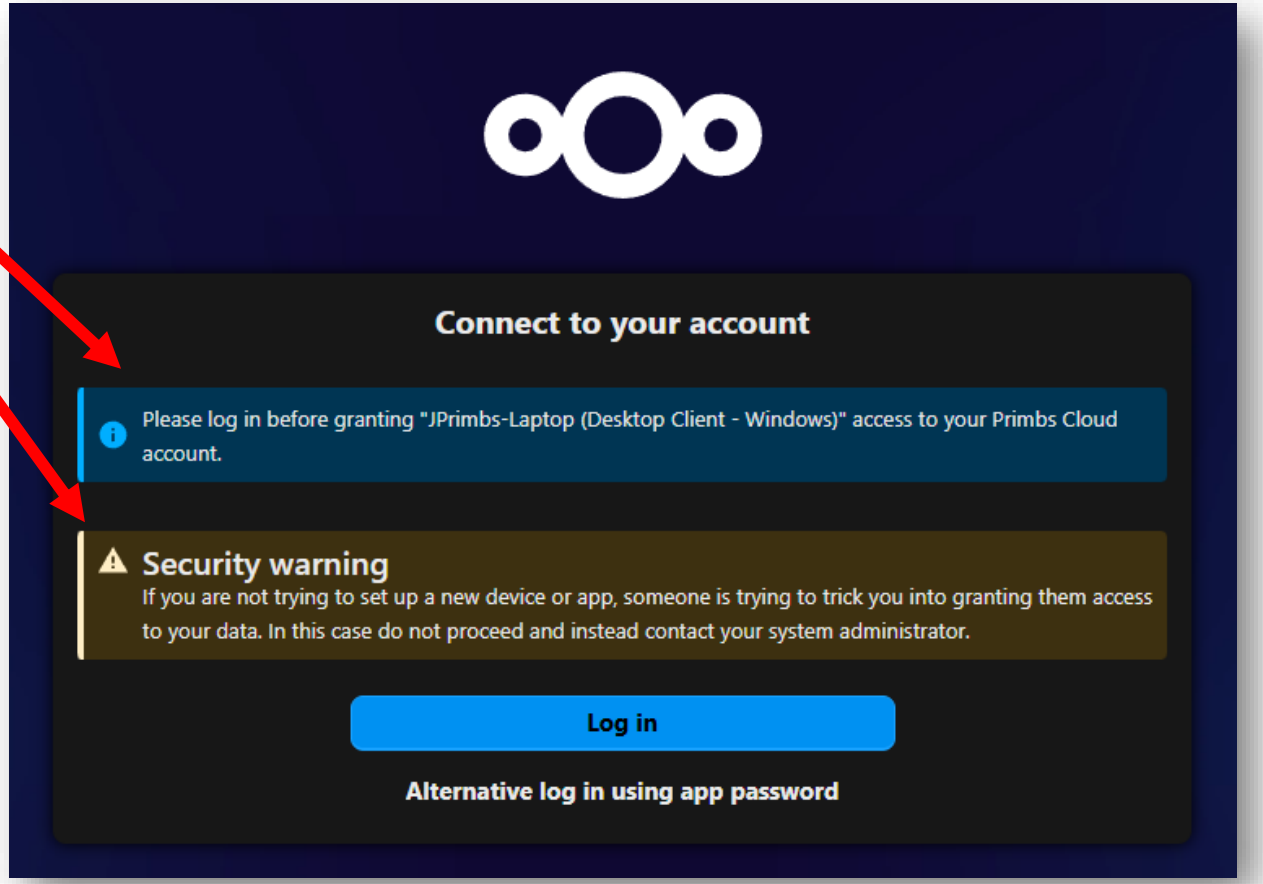


Source: https://pushsecurity.com/blog/consentfix

- **Solution**: Make clear to the RO what they authorize!

User Consent

- **Good example**: Nextcloud
  - Informs the RO what they are about to grant
  - Explicitly warns the user about the risks of granting access to client applications



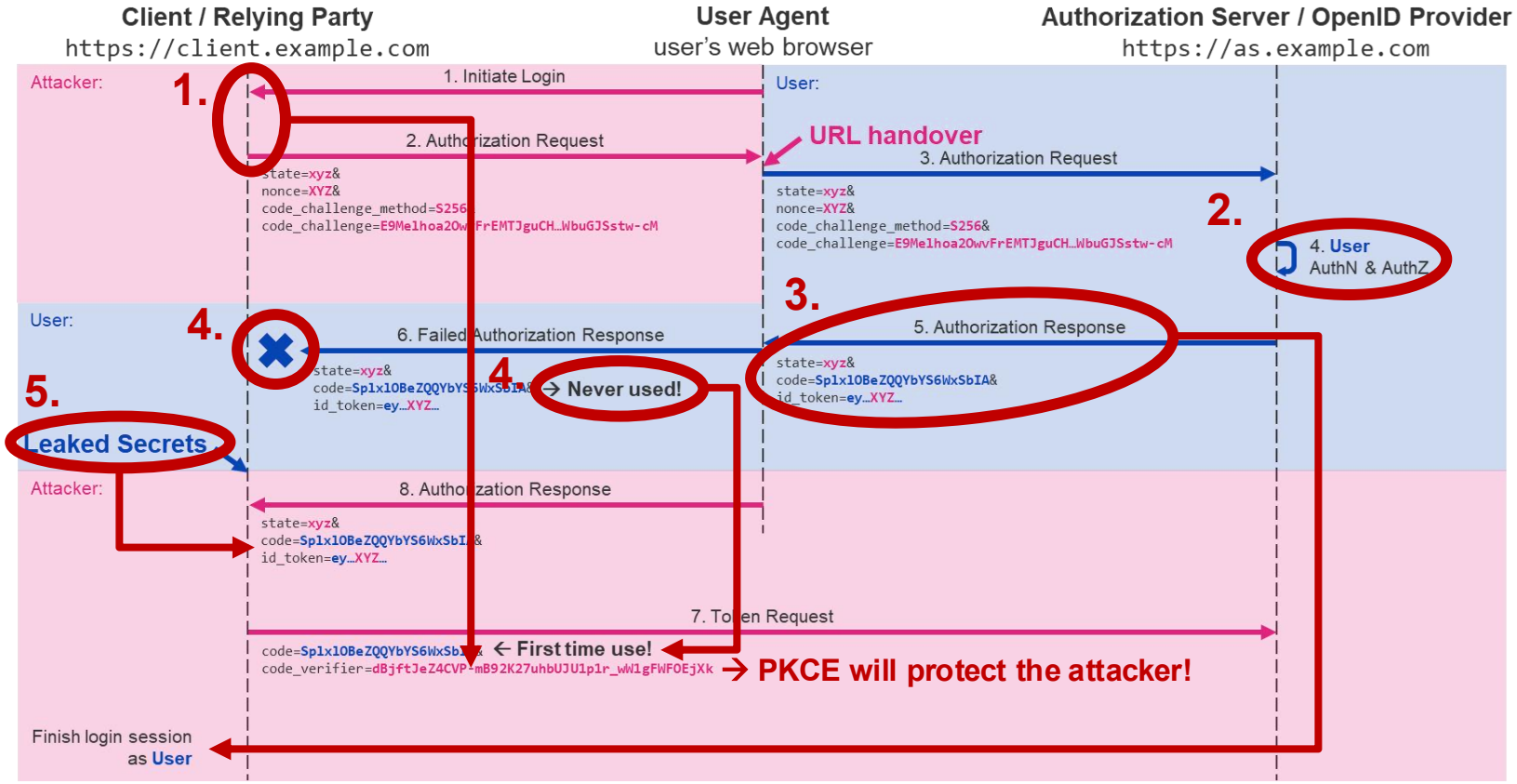
Conclusion

Conclusion: Why is Browser Swapping possible?

~~1. The attacker initiates the authorization flow~~

2. The user grants the authorization request
3. The AS/OP issues the user's authorization code with the attacker's state and nonce
4. The user's authorization code (one-time use) is not used by the user's Client/RP
5. The user leaks their authorization code to the attacker

► **Solving any of them fixes browser swapping!**



Conclusion

- **Long term solution**: HTTP Redirect Headers

- **Short-term solutions**
  - Reduce authorization code validity period as much as possible (5 seconds)
  - Prefer `response_mode=form_post` & enforce response mode
  - Ask user to consent & make clear to the users what they grant!
  - Otherwise: Remove authorization code from URL bar
    - Web: Retry login flow on failure or redirect to error page with tailing #
    - Mobile: Use custom URL scheme

The End

Thank you for listening!

`SameSite` Cookies with Form POST

- **Problem**: Sending a POST form cross-origin (from AS to client), browsers only send cookies with `SameSite=None`
  - Disables CSRF protection
- **Affected clients**
  - Backend Web Apps: yes
  - Frontend Web Apps: yes
  - Desktop Apps: no (no cookies needed)
  - Mobile Apps: no (no `form_post` applicable)
- **Solution**: Use temporary login session cookie (`oidc-session`) and use state as CSRF protection
  - Can be deleted after authorization response

- Backend Sessions:

SessionID	State	Nonce	CodeVerifier
abc	xyz	XYZ	dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk

- 2. Authorization Request:

```
HTTP/1.1 302 Found
Location: https://as.example.com/authorize?
  response_type=code%20id_token&
  redirect_uri=https%3A%2F%2Fclient.example.com%2Fcallback&
  client_id=s6BhdRkqt3&
  state=zyx&nonce=ZYX&
  code_challenge_method=S256&
  code_challenge=E9Melhoa2OwvFrEMTJguCH…WbuGJSstw-cM
Set-Cookie: oidc-session=abc; SameSite=None
```

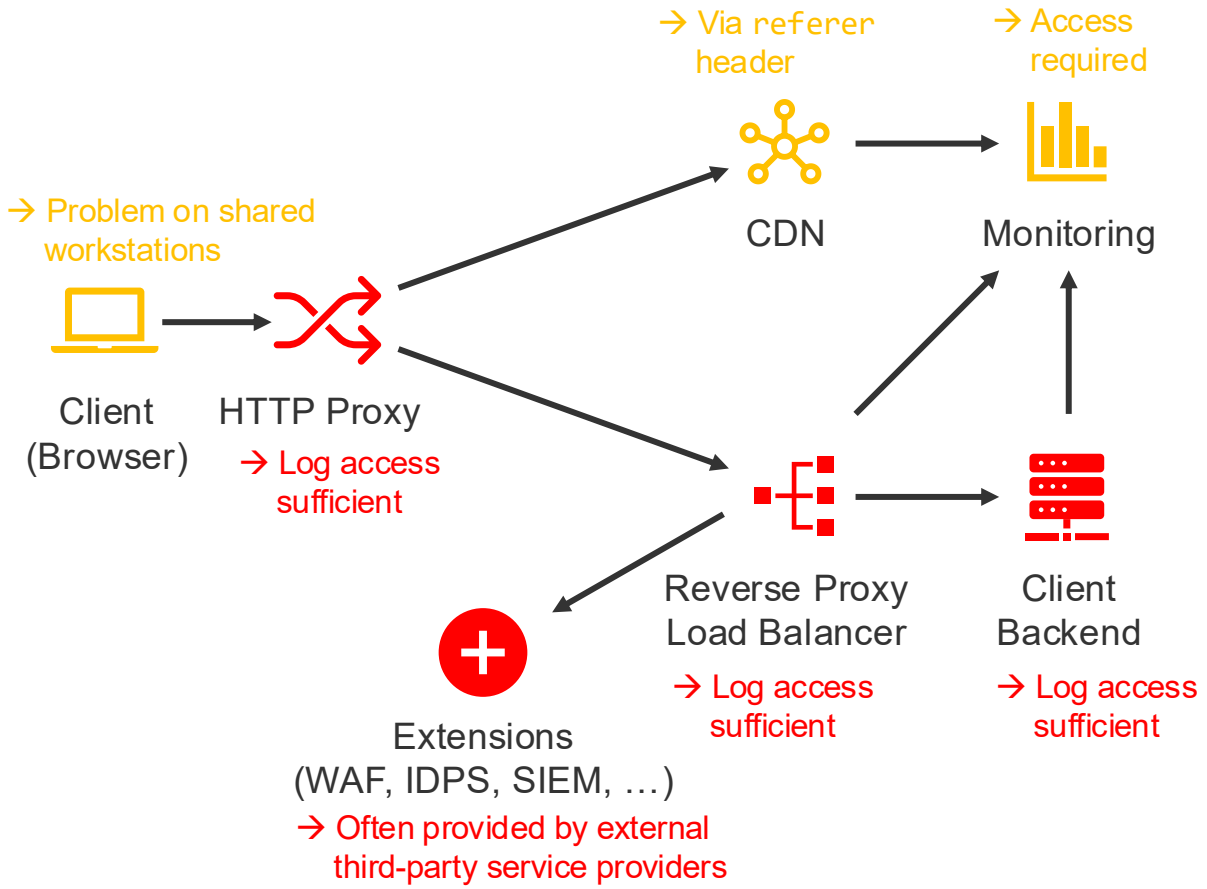
- 6. Authorization Response:

```
POST /callback HTTP/1.1
Cookie: oidc-session=abc
Content-Type: x-www-form-urlencoded

state=zyx&
code=SplxlOBeZQQYbYS6WxSbIA&
id_token=ey…ZYX…
```

Other Issues: Logging

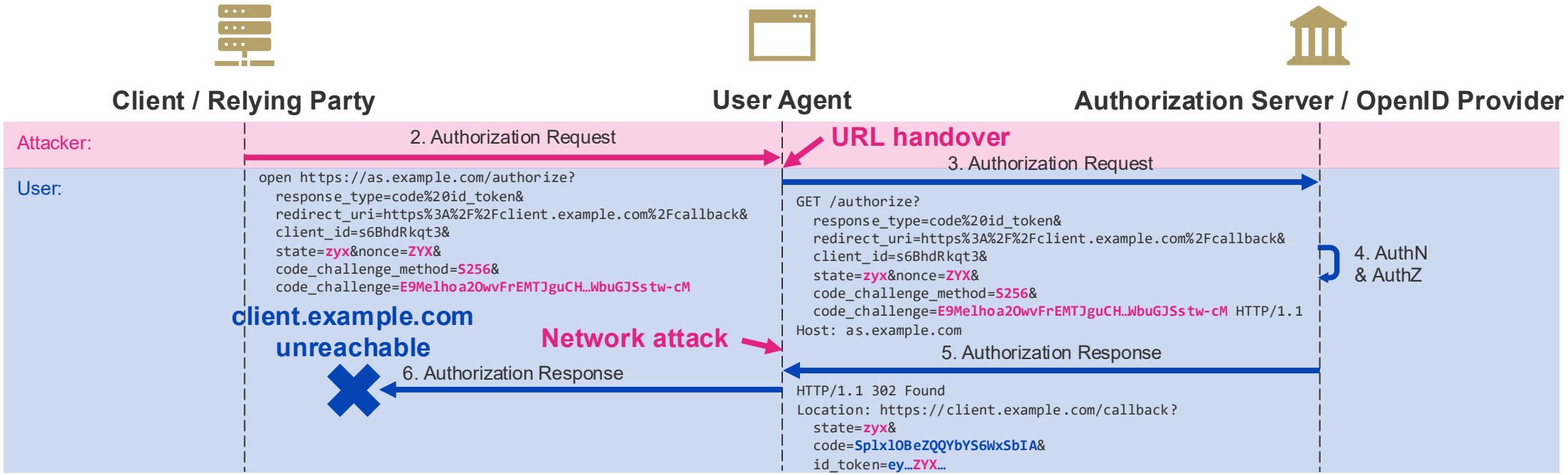
- **Problem:** Involved services (client, proxies, servers, middleware) typically log full URLs, including query parameters
  - Logs are reported to centralized monitoring systems
  - Everyone with near real-time access can access authorization codes from logs
- **Precondition**: Attacker has access to backend logs
  - Interesting for inside jobs, e.g., low-privileged log auditors attacking high-privileged administrators via the administration portal
- **Solution**: Log filtering to masquerade authorization codes → Service providers/administrators must know!



Unused OTS #3: Unreachable Client

Option 2: Network Attack

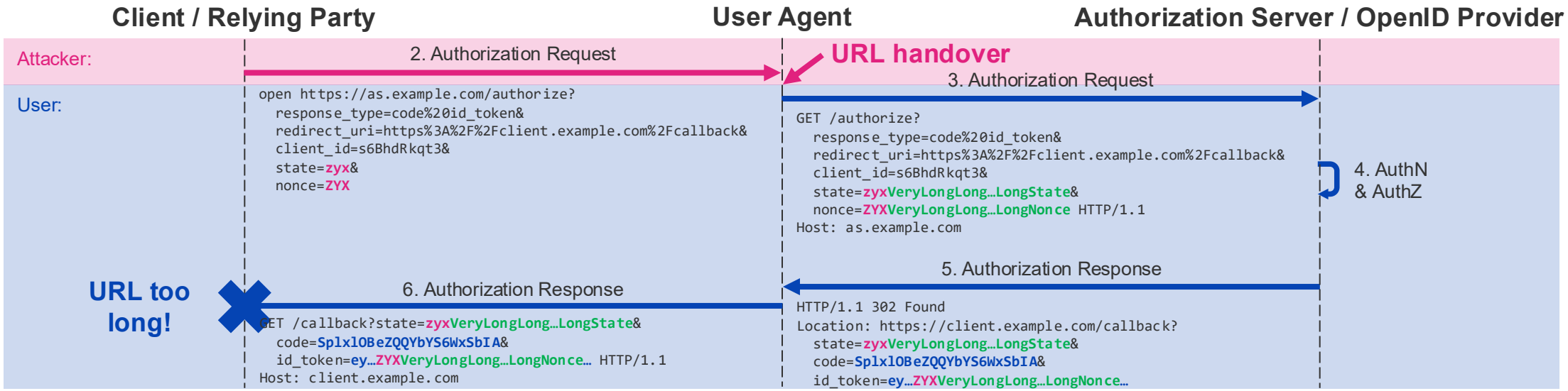
- **Issue**: Redirect URI may not be available
  - **Reasons**: (In)active VPN, firewall, ARP poisoning, DNS spoofing, rogue access points, …
- **Result:** Browser shows unreachable error; authorization code remains shareable in URL bar
- **Solution**: `response_mode=form_post` makes authorization code not sharable



Unused OTS #3: Unreachable Client

Option 3: Processing Issues

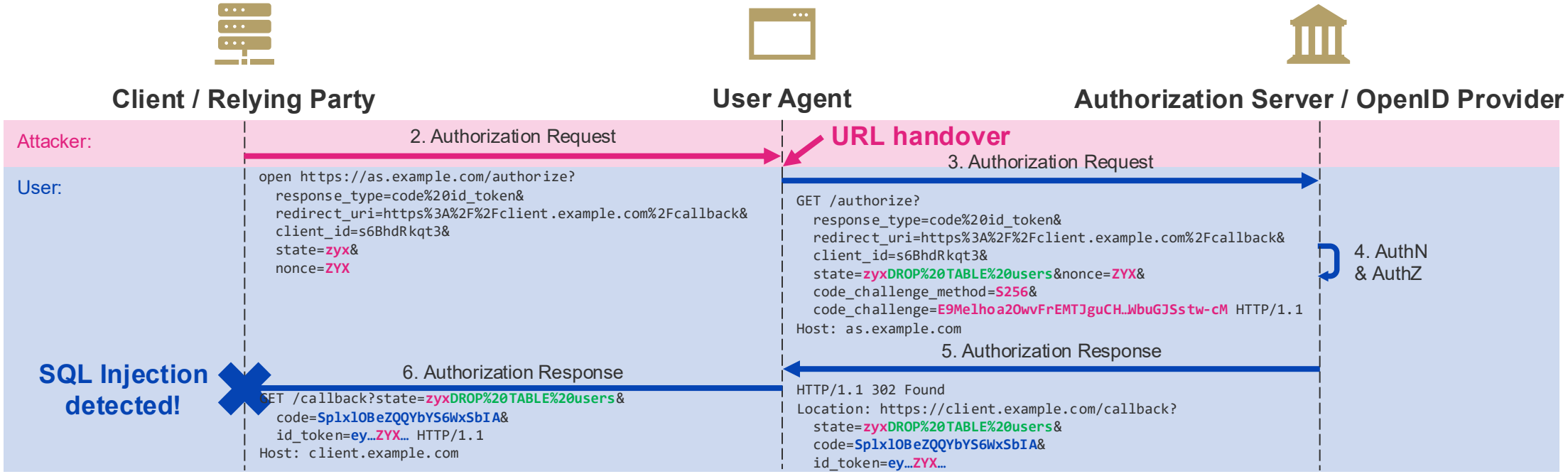
- **Issues**
  - Long URLs fail processing in browser or backend services
  - Attackers can influence URL length with state or nonce
- **Result:** User receives error from server; authorization code remains shareable in URL bar
- **Solution**: `response_mode=form_post` makes authorization code not sharable; post body typically has no length restrictions



Unused OTS #3: Unreachable Client

Option 4: Injection Protection

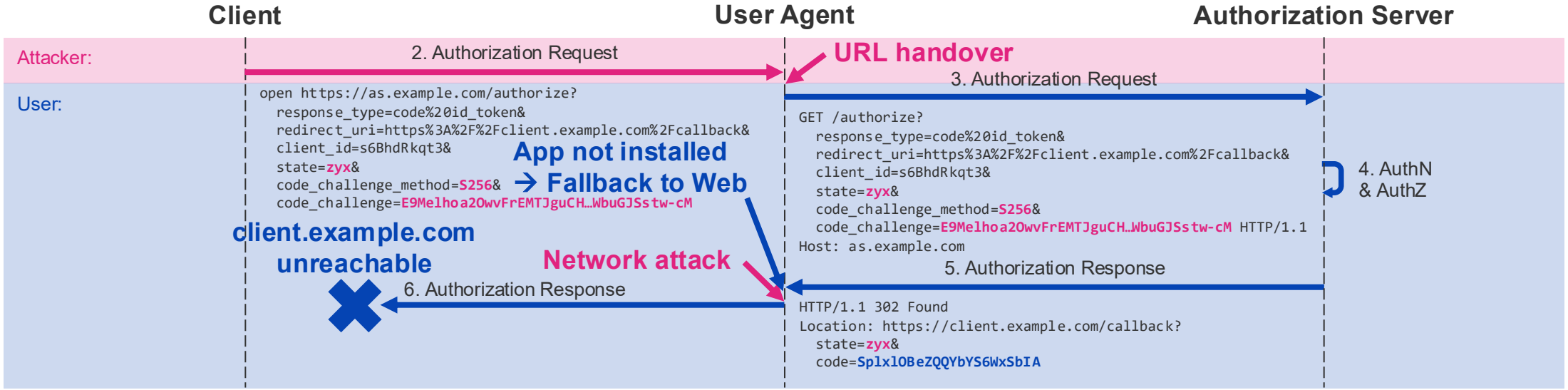
- **Issue**: Browsers and WAFs may not process URLs with a state value that is suspected of Cross-Site-Scripting (XSS), SQL-Injection, or similar attacks
- **Result:** Browser shows server error; authorization code remains shareable in URL bar
- **Solution**: `response_mode=form_post` makes authorization code not sharable



Unused OTS #3: Unreachable Client

Option 5: Missing Handler App

- **Issue**: Using Claimed HTTPS Redirect Scheme without the app installed on the victim's device, the app cannot be reached
- **Result**: The victim's device falls back calling the HTTPS server → see Option 2, 3, and 4
- **Solutions**:
  - ~~Use `response_mode=form_post`~~ → Not applicable for mobile apps
  - Have every app installed (**?!**)
  - Custom URL Schemes (**?**)



Other Solutions: Authorization Code Invalidation

- No standardized solution exists
- **Idea**: Invalid token request = authorization code invalidation
  - User's client cannot send a valid token request, if PKCE is used (only attacker knows the code verifier)
  - Some AS/OPs invalidate the authorization code if first token request contains invalid parameters, e.g.
    - Missing, empty or wrong `code_verifier` parameter
    - Missing, empty or wrong `redirect_uri` parameter
  - **Problem**: Heavily depends on AS/OP implementation; some AS/OPs ignore token request if not all parameters are correct → Authorization code remains valid

Other Solutions: PAR & JAR

- **Idea**: Protect authorization request with Pushed Authorization Request (PAR) or JWT-secured Authorization Requests (JAR)
  - Attacker cannot manipulate authorization request parameters
    - Mitigates response mode mutation
    - Enforces response mode even if AS/OP does not support response mode enforcement

- **Problems**
  - Only applicable to server-side OAuth clients (OIDC RPs or OAuth BFFs)
  - Only protects from authorization request manipulation (e.g., response mode mutation), but not from Browser Swapping itself
    - Attackers can still let the client generate a login flow in their session, but send the authorization request URL to the victim

Other Solutions: Automated Retries

- **Idea**
  - Legitimate user has a cookie session at the AS/OP
  - If the callback fails, the authorization request can be repeated before the user shares the authorization code

- **Problems**
  - Requires callback URL to be reachable
    - Example: redirect_uri=http://localhost:1234/
  - A valid authorization code still exists