Decoupled Flows in OAuth 2.0

An analysis of several of the newer "decoupled" profiles for OAuth 2.0 with particular emphasis on how the problem of session binding across devices is achieved.

Introduction

The flows that will be examined are:

- MODRNA Client Initiated Backchannel Authentication (CIBA) [1]
- OAuth 2.0 Device Flow [2]
- Standard Authorization Code Flow [3] with QR code & session bound to state parameter

Why are these flows needed

The CIBA spec usefully defines two types of devices:

- Consumption Device - the device on which the service that requires access to a user's resources, is accessed.
- Authentication Device - the device on which the user will authenticate to the Authorization server and authorize the client access to their resources.

Standard OAuth flows assume that the above 2 devices are the same, i.e. a user access a clients web application and accesses the authorization server in their browser or on their phone. This is why most OAuth 2 based implementations are based on the "redirect_uri". This parameter assumes that a user will access both the client and the authorization server on the same device.

There are several scenarios where this is not the case:

- A call centre worker requests access to a user's resources, a user remotely approves the request on their smartphone.
- A user wants to grant access to their resources to an input constrained or browserless device, e.g. a smart tv. The consumption device is the Smart TV, but the authentication device will be the user's laptop or smartphone.
- A user wants to pay for items at a POS terminal but authenticate and authorise the transaction on their smartphone.

Note that the CIBA spec has primarily been envisaged for usage by mobile network operators, but there is nothing inherent in the spec restricting it to such usage. In addition the Device Flow spec has primarily been aimed at input constrained devices that are in the resource owner's possession, but again I see a wider use-case.

[1] http://openid.net/specs/openid-connect-modrna-client-initiated-backchannel-authentication-1_0.html
[2] https://tools.ietf.org/html/draft-ietf-oauth-device-flow-07
[3] https://tools.ietf.org/html/rfc6749

Overview of the Flows

The following is a summary of the requests involved in the 3 flows. They are not exhaustive and I've left out standard OAuth 2.0 parameters for the sake of brevity. I've also used RP to mean "relying party" or client and OP to mean "OpenID Provider" or Authorization Server.

MODRNA Client Initiated Backchannel Authentication (CIBA)

1. RP sends Backchannel Authentication Request to `/bc_authorize`, with:
    a. `login_hint`
    b. `binding_message` (optional)
    c. `scope`
2. OP responds with Backchannel Authentication Response containing:
    a. `auth_req_id`
3. OP obtains end-user consent/authorization. This process will normally start with a push notification or similar. If a binding message was sent, the OP must show the end-user the binding message and the user must confirm that it is the same as the binding message displayed on the consumption device.
4. RP polls `/token` endpoint with:
    a. `grant_type: "urn:openid:params:modrna:grant-type:backchannel_request"`
    b. `auth_req_id`
5. OP responds with tokens

OAuth 2.0 Device Flow

1. RP sends Device Authorization Request to `/device_authorization`, with
    a. `scope`
2. OP responds with Device Authorization Response containing:
    a. `device_code`
    b. `user_code`
    c. `verification_uri`
3. User navigates to verification uri on their authentication device, authenticates, enters the `user_code` and grants consent.
4. RP polls `/token` endpoint with:
    a. `grant_type: "urn:ietf:params:oauth:grant-type:device_code"`
    b. `device_code`
5. OP responds with tokens

Standard Authorization Code Flow with QR code & session bound to state parameter

1. RP generates standard authorization request and displays as QR code. This authorization request has a signed state parameter for the particular session

2. User navigates to the authorization endpoint on their authentication device, authenticates and grants consent
3. User is redirected to the RP's `redirect_uri`, and a standard authorization code token grant is performed (NB this is performed without the user having a "logged-in" session with the RP).
4. RP's consumption device polls RP's servers to find when a valid token has been granted against the issued state parameter

Trade Offs

CIBA can be initiated with no user interaction. For this to work the RP must be in possession of a valid user identifier. This is not the case in most other OAuth 2 based flows (although it is possible for such an identifier to be provided in OIDC flows).

When implemented with appropriate safeguards, the backchannel initiated flow can provide an excellent user experience. It is not widely used however and its threat model is not as well understood as standard "redirect" based flows.

The other two flows require the user to initiate the flow. This restricts their use case, but makes them more similar to standard redirect flows. The fact that 2 devices are involved means that the OP cannot rely on 'fraud markers" in the redirected user-agent as they currently do in standard redirect flows.

The third flow that I refer to requires extra work to be performed by the RP out of the scope of the core specifications. This could lead to insecure implementations without the appropriate peer-reviewed discussion of security considerations.

Session Binding

By session binding I'm referring to ensuring that the consumption device that is being granted access is the intended consumption device by the user of the authentication device (i.e. the resource owner).

For many of the use-cases, this can be expressed in simpler terms by saying that the same user is accessing the consumption device and the authentication device. However the use-case of the remote call centre agent doesn't fit into that description.

Redirect-based OAuth flows have strong session binding characteristics when the user stays on the same device. OAuth 2 supports the state parameter. OIDC added in nonce - with its playback in the id_token. For mobile apps, the PKCE extension further helps binds sessions.

Decoupled flows have to solve session binding in different ways however. To compare the different flows characteristics I will look at these two attacks:

- Same consumption device, different user - this is where an attacker hijacks the session to authenticate themselves on the consumption device and is referred to as “session spying”.
- Different consumption device , same user - this is where an attacker tricks the user into authorising access to a different consumption device

I will also examine the confidence the RP has in the integrity of the session and the confidence the OP has in the integrity of the session.

Session Binding in CIBA:

Same consumption device, different user. This attack is difficult in CIBA itself as it is the OP’s responsibility to initiate the authentication session with the user. Depending on the type of `login_hint`, this could be possible, but it is outside of the scope of the CIBA spec.

Different consumption device , *same user*. In CIBA there is an optional binding_message. Without this it would be possible for a malicious actor to trick the user into authorising access to the wrong consumption device. For example, an attacker could start a CIBA flow just before the user requested that a legitimate flow start. When the user authenticates on their authentication device, they would have no way of knowing that they were actually authorising the attackers session and not their own.

CIBA, therefore supports the use of a binding message. This is generated by the RP and passed to the OP who must display it to the user on the authentication device. The inference from the spec is that the user should confirm that the same binding message is shown on both the consumption device and the authentication device.

For higher value scenarios, the user could be asked not to visually compare, but rather to enter the binding message shown on the consumption device into the authentication device. The OP could then compare the binding message sent by the RP and the one entered by the user.

Because the nature of CIBA means that the user doesn’t explicitly start a session, there is an inherent risk that an attacker initiates a CIBA flow to confuse the user. Even if the RP sends a binding_message it has to rely on the OP to ensure the binding_message is confirmed by the user.

CIBA with a binding_message therefore gives the OP confidence in the session’s integrity. The RP however has to trust the OP; for many scenarios this is acceptable, but there are scenarios where it would be beneficial for the protocol to give both the OP and RP assurance of the integrity of the session.

The fact that the binding_message isn’t required to be entered by the user is a weakness in the protocol. Asking the user to visually compare a binding_message on two devices is too error prone a mechanism to rely on.

Session Binding in Device Flow

In the Device Flow, there are 2 parameters used to help bind the session across devices,, the user_code and the device_code. The user_code is somewhat analogous to the binding_message in CIBA, except that it is generated by the OP not the RP.

The use of these two codes (assume rate-limiting is implemented to prevent brute-forcing) gives the following characteristics.

Same consumption device, different user. This attack is possible and is called out in the security considerations of the device flow spec. A malicious user just needs to see the user_code displayed on the consumption device and they can hijack the session. For many scenarios this is an acceptable risk that can be mitigated in other ways, for example, showing an informative error message when the real user tries to authenticate.

Different consumption device , same user. This attack is not possible in the device flow as there is a hidden device_code that is part of the protocol and this is bound to the user code. There are social engineering attacks possible however and these are called out in the security considerations as "remote phishing". Because of the possibility of such attacks the specification recommends that the user enters the user_code rather than having it prefilled. This is analogous to requiring the CIBA user to enter the binding_message.

The design of the Device Flow protocol is such that all assurance about the integrity of the session is with the OP and not the RP. This is an acceptable trust model for most scenarios.

Session Binding in Standard Authorization Code Flow with QR code & session bound to state parameter

This flow makes use of the standard authorization code flow however the fact that 2 devices are used gives it the following characteristics with regards to session binding:

Same consumption device, different user. This attack is possible in the same way that it is possible in the device flow. An attacker who can view the QR code that encodes the redirect_uri can hijack the session. There are however mitigations possible in this flow that are not possible in the device flow. Because the RP has interaction with the user on the authentication device, there is a possibility of asking some further confirmation questions of the user before exchanging the authorization code for an access token.

Different consumption device , same user. This attack can be protected at the protocol level by using extensions such as PKCE or the nonce in OIDC. However, the social engineering attack that is possible for the device flow is again possible with this flow. To protect against this attack the RP has the same opportunity to ask confirmation questions on the authentication device before fetching the access token.

The advantage that this flow has over device flow is that it gives the RP more opportunities to gain an acceptable level of assurance that the right user has granted access to the intended device.

Conclusion

Decoupled flows will become more common, so it is important that the OAuth community examine such flows to highlight security considerations.

Each of the flows examined as positives and negatives and each one is suitable for certain scenarios. I have a particular interest in the CIBA flow as it can potentially meet the requirements of the European Banking Authority's regulatory technical standards for strong customer authentication and common and secure open standards of communication. These standards prohibit banks from only offering redirect based flows when providing API access to payment accounts.

I plan to propose an extension to CIBA that will give the RP more assurance of session integrity. I also believe that there needs to be documentation for alternative methods of confirming the binding_message.