Open Created 3 weeks ago by Sascha Preibisch

Consent Receipt

WIP

The concept of a consent receipt associated with a consent management system becomes more prominent these days. The idea of a consent receipt was first mentioned during an IIW (Internet Identity Workshop) event in October 2015. A consent receipt mainly represents a digital copy of a receipt as issued in a supermarket. Since then Identity Management Platforms (IDP) are introducing this concept.

Use case

Specifically in flows where users (resources owners) give consent to share personal data or approve a transaction and similar these users do not have any type of proof that shows what they have consented to. Some IDM do list granted applications and associated (OAuth) scope/permissions but that is not sufficient.

However, when looking at OAuth and its derived protocols, there is a lot of data to capture during an authorization:

- client name/ client_id
- relying parties ip address, name
- scope / permissions
- transaction details
- date / time
- authorization server ip address, name, provider
- current terms and conditions of the authorization server and the relying party
- current privacy statement of the authorization server and the relying party
- given username/ userId
- consent decision (grant/deny, yes/no, approve,disapprove)
- acr (authentication context class reference)
- amr (authentication method reference)

Receipt

In any case, a user always has the option to grant or deny consent. No matter what the user chooses, a consent receipt needs to be issued. For **Deny** cases a simpler receipt may be sufficient.

The receipt is created by the party that receives the consent decision, typically the authorization server. The document structure is JSON, and it is represented as a JSON Web Token (JWT) and signed using the authorization server's private key.

The receipt is made available to the user just-in-time, via user dashboard and APIs.

Phase 01

To get started with a consent receipt feature, which may evolves into a consent management solution in the future, it is important to be prepared for different ways of creating and consuming receipts. Right from the beginning APIs need to allow CRUD operations.

The first integration would be used by ourselves. A *click* on **Grant** or **Deny** at our own OIDC authorization server could be the first consumer of consent receipts APIs.

The authorization server receives the consent decision and generates a JSON based receipt. It turns it into a JWT using its private signing key.

The authorization server sends the JWT to the consent services **/receipt** endpoint. The server authenticates using an API key.

APIs

APIs are at the heart of this feature and provide CRUD operations. All APIs are protected and require an authorized client.

Authorization

Authorization for these APIs is supported via API Keys and OAuth.

NOTE: API Keys are only supported for our own authorization server. API Keys are LoginID serviceToken and have to be signed by LoginID.

The receipt APIs support the following SCOPEs:

- `receipt:list` : used to list receipts
- `receipt:create` : used to create a new receipt that uses the state `active`
- `receipt:revoke` : used to revoke given consent. This results in a new receipt that references the original one and uses the state `revoked`
- `receipt:delete` : removes a receipt from the system

Create API

A receipt is created as JSON payload and needs to have the following content:

```
{
  "relying_party": {
    "client name": "",
    "client_id": "",
    "ip_address": "",
    "organization": "",
    "redirect_uri": "",
    "href": {
      "terms_of_service": "",
      "privacy_statement": ""
    }
  },
  "transaction": {
    "permissions": [],
    "date": 1234567890
  },
  "issuer": {
    "iss": "",
    "authorization_endpoint": "",
    "token_endpoint": "",
    "ip_address": "",
    "organization": "",
    "href": {
      "terms_of_service": "",
      "privacy_statement": ""
    }
  },
  "subject": {
    "username": "",
    "consent": "",
    "acr": "",
    "amr": ""
  },
  "id": ""
}
```

That payload has to be signed by the issuer of the receipt and submitted to the consent service `/receipts` API:

- `Method` : POST
- `Path` : /receipts
- `Content-Type` : application/json
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:create`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:create`
- `Body` :

```
{
  "receipt": "{jwt}"
}
```

The API verifies the access_token and checks for the required SCOPE.

The API then generates a **receiptId**, **status** and **created** timestamp. The JWT and these values are persisted in the service's database. The API also extracts the associated username ($.subject.username) and clientId ($.relying_party.client_id) so that queries can search for those values.

The API response looks like this:

- `HTTP Status` : 201 (Created)
- `Content-Type` : application/json
- `Body` :

```
{
  "receiptId": "",
  "status": "",
  "created": 1234567890
}
```

Retrieve API

Consent receipts may be retrieved at the same `/receipts` API:

Retrieving many receipts:

- `Method` : GET
- `Path` : /receipts
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:list`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:list`

Retrieving receipts with query components (using one or many):

- `Method` : GET
- `Path` : /receipts?userId={userId}&clientId={clientId}
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:list`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:list`

Retrieving one single receipt:

- `Method` : GET
- `Path` : /receipts/{receiptId}
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:list`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:list`

Delete API

Consent receipts may be deleted at `/receipts` API:

Delete a receipt for userId, clientId and optionally the status:

- `Method` : DELETE
- `Path` : /receipts?userId={userId}&client_id={clientId}&status={status}
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:delete`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:delete`

Delete a single receipt:

- `Method` : DELETE
- `Path` : /receipts/{receiptId}
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:delete`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:delete`

Revoke API

In comparison to deleting a receipt, this request is more or less a replacement of an existing one. For example, if a client has received consent for `scope=A B C` but now requests `D` in addition (`scope=A B C D`), a new consent is needed and therefore a new receipt. Existing receipts can be matched by userId + clientId. The API finds the receipt with status `active` and sets t to `revoked` . The new receipt witll have status `active` . It is important to keep the history of receipt to show a user how the active consent was achieved.

Consent receipts may be revoked at `/receipts` API:

Revoke (replace) an existing receipt:

- `Method` : PUT
- `Path` : /receipts
- `Authorization` : Bearer {access_token} // issued for scope = `receipt:revoke`
  - `Authorization: APIKey {loginid-serviceToken}` // issued for scope = `receipt:revoke`
- `Content-Type` : application/json
- `Body` :

```
{
  "receipt": "{jwt}"
}
```

Edited 2 weeks ago by Sascha Preibisch

0 0 0 Show labels

Sascha Preibisch @sascha17 added Idea: New label 3 weeks ago

Sascha Preibisch @sascha17 changed the description 3 weeks ago ·

Sascha Preibisch @sascha17 changed the description 2 weeks ago ·