RECOMMANDATIONS POUR LA SÉCURISATION DE LA MISE EN ŒUVRE DU PROTOCOLE OPENID CONNECT

GUIDE ANSSI

ANSSI-PA-080
08/09/2020

PUBLIC VISÉ :

Développeur Administrateur RSSI DSI Utilisateur

Informations

Attention

Ce document rédigé par l'ANSSI présente les **« Recommandations pour la sécurisation de la mise en œuvre du protocole OpenID Connect »**. Il est téléchargeable sur le site `www.ssi.gouv.fr`.

Il constitue une production originale de l'ANSSI placée sous le régime de la « Licence ouverte v2.0 » publiée par la mission Etalab [**?**].

Conformément à la Licence Ouverte v2.0, le guide peut être réutilisé librement, sous réserve de mentionner sa paternité (source et date de la dernière mise à jour). La réutilisation s'entend du droit de communiquer, diffuser, redistribuer, publier, transmettre, reproduire, copier, adapter, modifier, extraire, transformer et exploiter, y compris à des fins commerciales.

Ces recommandations n'ont pas de caractère normatif, elles sont livrées en l'état et adaptées aux menaces au jour de leur publication. Au regard de la diversité des systèmes d'information, l'ANSSI ne peut garantir que ces informations puissent être reprises sans adaptation sur les systèmes d'information cibles. Dans tous les cas, la pertinence de l'implémentation des éléments proposés par l'ANSSI doit être soumise, au préalable, à la validation de l'administrateur du système et/ou des personnes en charge de la sécurité des systèmes d'information.

Évolutions du document :

VERSION	DATE	NATURE DES MODIFICATIONS
1.0	08/09/2020	Version initiale

Table des matières

1 Introduction

L'administration, les entreprises publiques et privées offrent de plus en plus de services accessibles par Internet tels que les téléservices pour les démarches administratives ou la souscription d'offres commerciales. Ces fournisseurs de services mettent à disposition des usagers de nombreuses données stockées sous forme dématérialisée. Ces entités simplifient leurs accès :

- en centralisant l'authentification des utilisateurs grâce à des fournisseurs d'identité qui possèdent de manière massive des identités ;
- en facilitant les échanges d'informations (justificatif de domicile, revenu fiscale de référence, photos, etc) entre elles.

Le protocole OpenID Connect normalise les échanges pour réaliser ces deux fonctions. Ce guide rappelle d'abord brièvement le protocole dans le chapitre 2.2 et fournit ensuite des recommandations sur sa mise en œuvre dans le chapitre 3. Le document ne décrit pas entièrement le protocole dont la documentation est disponible sur le site internet OIDC[1].

1.1 Dénominations

- API : L'Application Programming Interface, une interface de programmation applicative décrivant les méthodes et paramètres acceptés par une application.
- Certificat pinning : technique qui consiste à associer dans une application un certificat à un serveur distant soit en utilisant le certificat lui même, soit son empreinte, soit sa clé publique. Le but est de comparer les informations préchargées dans l'application avec celles reçues lors des échanges TLS et se prémunir contre un serveur ayant une fausse identité.
- Claim : information sur un utilisateur détenue par une entité.
- Cinématique : ensemble de communications réseau entre plusieurs acteurs.
- CSRF : Cross Site Request Forgery, une attaque permettant de faire exécuter à un utilisateur une action à son insu lorsqu'il est connecté à une application.
- HTTP : L'HyperText Transfert Protocol, un protocole de communication client serveur développé pour Internet.
- HTTPS : L'HyperText Transfert Protocol Secure, le protocole HTTP sécurisé au travers de tunnels TLS.
- HTML : L'HyperText Markup Language, un langage de balisage utilisé par les navigateurs pour représenter les pages d'un site Web.
- HMAC : Hash-based Message Authentification Code, un code d'authentification reposant sur une empreinte calculée à l'aide d'une fonction de hachage et d'une clé.

- JSON : JavaScript Object Notation, un format de données textuel permettant de représenter une information structurée.
- JWT : JSON Web Token, standard défini par la RFC 7519[10] permettant d'échanger de l'information sur une entité au format JSON.
- JWE : JSON Web Encryption, standard défini par la RFC 7516[12] permettant de protéger un JWT en confidentialité.
- JWS : JSON Web Signature, standard défini par la RFC 7515[9] permettant de protéger un JWT en intégrité par un HMAC ou une signature.
- OAuth2.0 : cadre, défini par la RFC 6749[7], décrivant des mécanismes pour obtenir l'autorisation d'accéder à des ressources protégées.
- OIDC : OpenID Connect, protocole reposant sur OAuth2.0 pour définir les échanges permettant de récupérer l'identité d'un utilisateur authentifié.
- OWASP : Open Web Application Security Project, une organisation non lucrative définissant des bonnes pratiques de sécurité des applications Web.
- Redirection ouverte : redirection non vérifiée d'un utilisateur. Cette vulnérabilité peut permettre de voler des informations sensibles.
- RGS : Référentiel Général de Sécurité.[6]
- TLS : Transport Layer Security, un protocole de sécurisation des communications réseaux.
- URL : L'Uniform Resource Locator, l'adresse Web d'une ressource (HTML, API, etc.).
- XSS : Cross Site Scripting, une attaque consistant à injecter du code malveillant afin qu'il soit exécuté par le navigateur d'un utilisateur.

2
Qu'est ce que OpenID Connect

Ce chapitre a pour objectif de rappeler le fonctionnement de la norme OIDC et de définir les notions nécessaires à la compréhension des recommandations de sécurité.

La norme OIDC [1] définit les échanges pour mettre en œuvre un point d'authentification unique et des services de partage d'informations. Elle s'appuie sur le standard OAuth2.0 [7] qui définit des mécanismes permettant à une application tierce d'obtenir le consentement de l'utilisateur pour accéder temporairement à ses ressources tout en protégeant ses authentifiants.

La norme OIDC complète le protocole OAuth2.0 :

- en ajoutant l'ID token, un JWT [10] contenant des informations sur l'authentification de l'utilisateur ;
- et en normalisant la récupération d'informations sur l'utilisateur.

Les mécanismes d'authentification d'un utilisateur ne sont pas définis par la norme OIDC et ne sont pas abordés dans ce document.

Le schéma 2.1 décrit de manière simplifiée les échanges entre les acteurs.

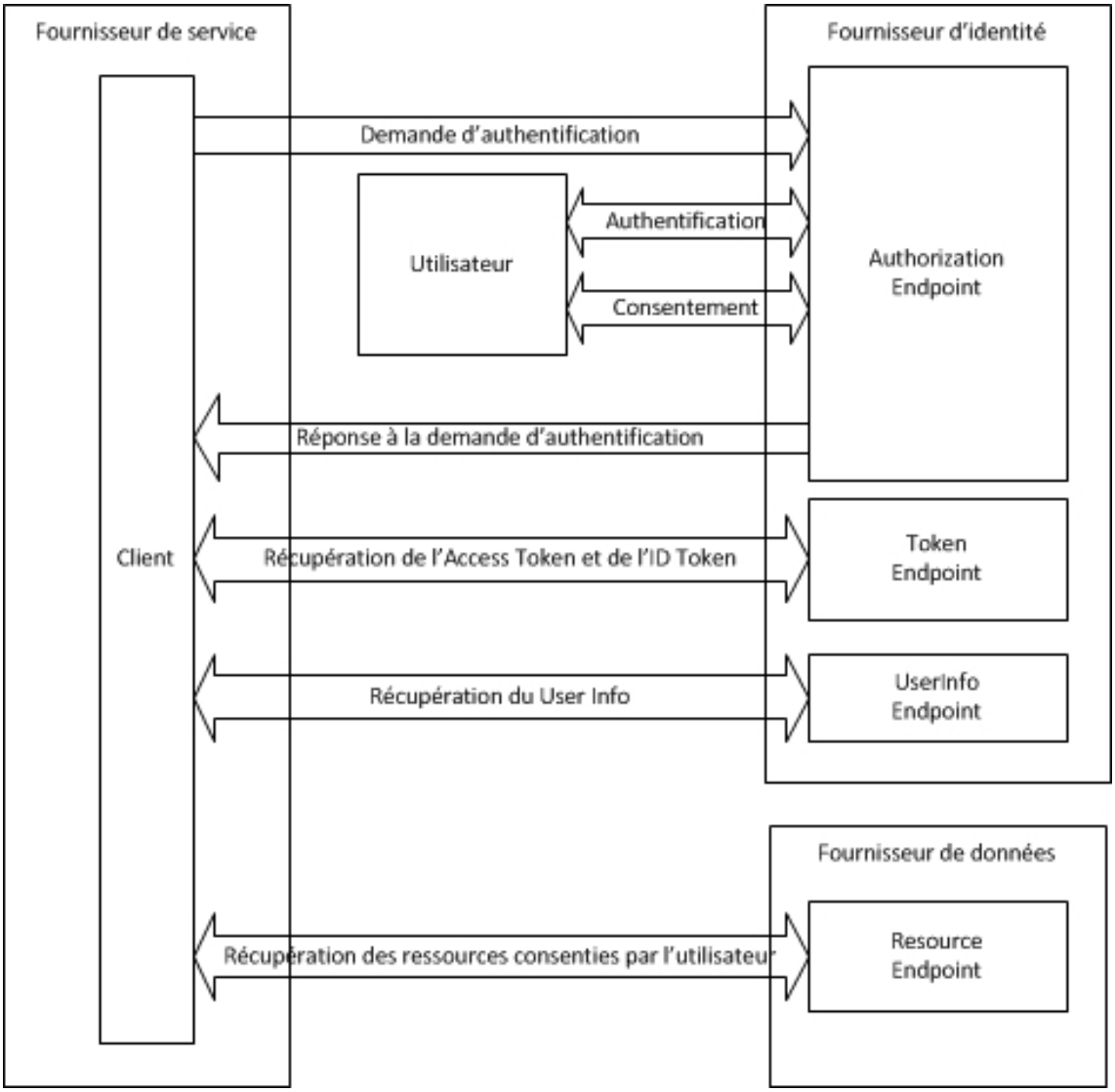


FIGURE 2.1 – Schéma simplifié des échanges entre les différents acteurs

2.1 Les acteurs

Dans le schéma 2.1, on retrouve les acteurs suivants :

- **l'utilisateur :** il souhaite accéder à une information ou un service ;
- **le fournisseur de service :** il a besoin d'identifier un utilisateur et d'accéder à certaines de ses ressources. Celles-ci peuvent être détenues par un autre fournisseur de service ou un fournisseur de données. Il existe plusieurs types de client OIDC :
  - **confidentiel :** le client est opéré dans un environnement maîtrisé (serveur). Les secrets d'authentification (mot de passe, clé privée) fournis lors de l'enrôlement peuvent être protégés, les requêtes en provenance de ce type de client peuvent être authentifiées,
  - **public :** le client (application native, client léger) est opéré dans un environnement généralement non maîtrisé (téléphone mobile, poste utilisateur). La protection des secrets d'authentification est difficile à mettre en œuvre, les requêtes de ce type de client ne peuvent donc pas être authentifiées de manière fiable ;
- **le fournisseur d'identité :** il met en œuvre trois services :
  - un ***Authorization Endpoint***, ce service authentifie l'utilisateur et obtient son consentement. En retour, il fournit au client OIDC un *authorization code*,
  - un ***Token Endpoint***, ce service permet au client OIDC de récupérer un *access token* et un *ID Token* en échange d'un *authorization code*,
  - un ***UserInfo Endpoint***, ce service permet au client OIDC de récupérer le *UserInfo*, des informations sur l'utilisateur authentifié (prénom, nom, adresse, …), en échange d'un *access token* ;
- **le fournisseur de données :** met en œuvre un ***Resources Endpoint***. Ce service permet au client OIDC, grâce à un *access token*, de récupérer des ressources dont l'accès a été consenti par l'utilisateur, après ou lors de la demande d'authentification.

2.2 Les principaux éléments échangés

Selon les cinématiques proposées par le protocole, les éléments suivants sont échangés :

- **les *Claims*** : des informations sur l'utilisateur et son authentification ;
- **l'*authorization code* :** un code généré par l'*Authorization Endpoint* suite à une authentification ;
- **l'*access token* :** généré par le *Token Endpoint*, il permet de récupérer des *Claims* sur l'utilisateur dont le *UserInfo* ;
- **l'*ID Token* :** un JWS, généré par le *Token Endpoint*, contenant des *claims* sur l'authentification de l'utilisateur ;
- **le *UserInfo*** : un JWT ou un JWS, généré par le *UserInfo Endpoint*, contenant des *claims* sur l'utilisateur (nom, prénom, adresse, numéro de téléphone, …).

2.3 Les cinématiques OIDC

OIDC définit trois types de cinématique d'authentification :

- **la cinématique *authorization code* :** est mise en œuvre pour des clients OIDC de type confidentiel. Elle utilise les *authorization code* et les *access token* (voir schéma 3.1). Elle est définie par le protocole OAuth2.0. ;
- **la cinématique *implicit* :** est mise en œuvre pour des applications clientes OIDC dites publiques et repose uniquement sur les *access token* (voir annexe A). Elle est définie par le protocole OAuth2.0. ;
- **la cinématique *hybrid* :** est un mélange des deux cinématiques précédentes. Non définie dans le protocole OAuth2.0, elle a été introduite par la norme OIDC.

R1

Utiliser en priorité la cinématique authorization code pour les applications Web

Il est recommandé d'utiliser la cinématique *authorization code* pour les applications Web. Elle permet :

- de protéger en confidentialité l'*access token* de l'environnement de l'utilisateur ;
- de mettre en œuvre un mécanisme d'authentification entre le client OIDC et le *Token Endpoint*.

R2

Mettre en place une politique de sécurité

Il est recommandé de mettre en place une politique de sécurité définissant les accès aux informations des fournisseurs de données. Cette politique pourrait comprendre, par exemple, une durée de validité différente pour les *access token* en fonction de la sensibilité des informations accédées et de la cinématique utilisée.

Le protocole OAuth2.0 propose deux cinématiques supplémentaires qui ne sont pas reprises par la norme OIDC. Seule la cinématique *authorization code* sera abordée dans la suite de ce document.

3
Sécuriser la mise en œuvre de la cinématique authorization code

L'objectif de ce chapitre est de fournir des recommandations de sécurité lors des différentes phases d'échanges de la cinématique *authorization code*.

La cinématique *authorization code* est choisie par le fournisseur de service au moment de l'enrôlement d'un client OIDC de type confidentiel auprès d'un fournisseur d'identité (voir §2.1). Les secrets qui authentifient leurs requêtes peuvent être stockés de manière sécurisée. Le fournisseur de service aura été préalablement enrôlé afin de lui fournir le paramétrage nécessaire à la mise en service de ses clients OIDC. Les échanges entre les différents acteurs sont définis dans le schéma 3.1.

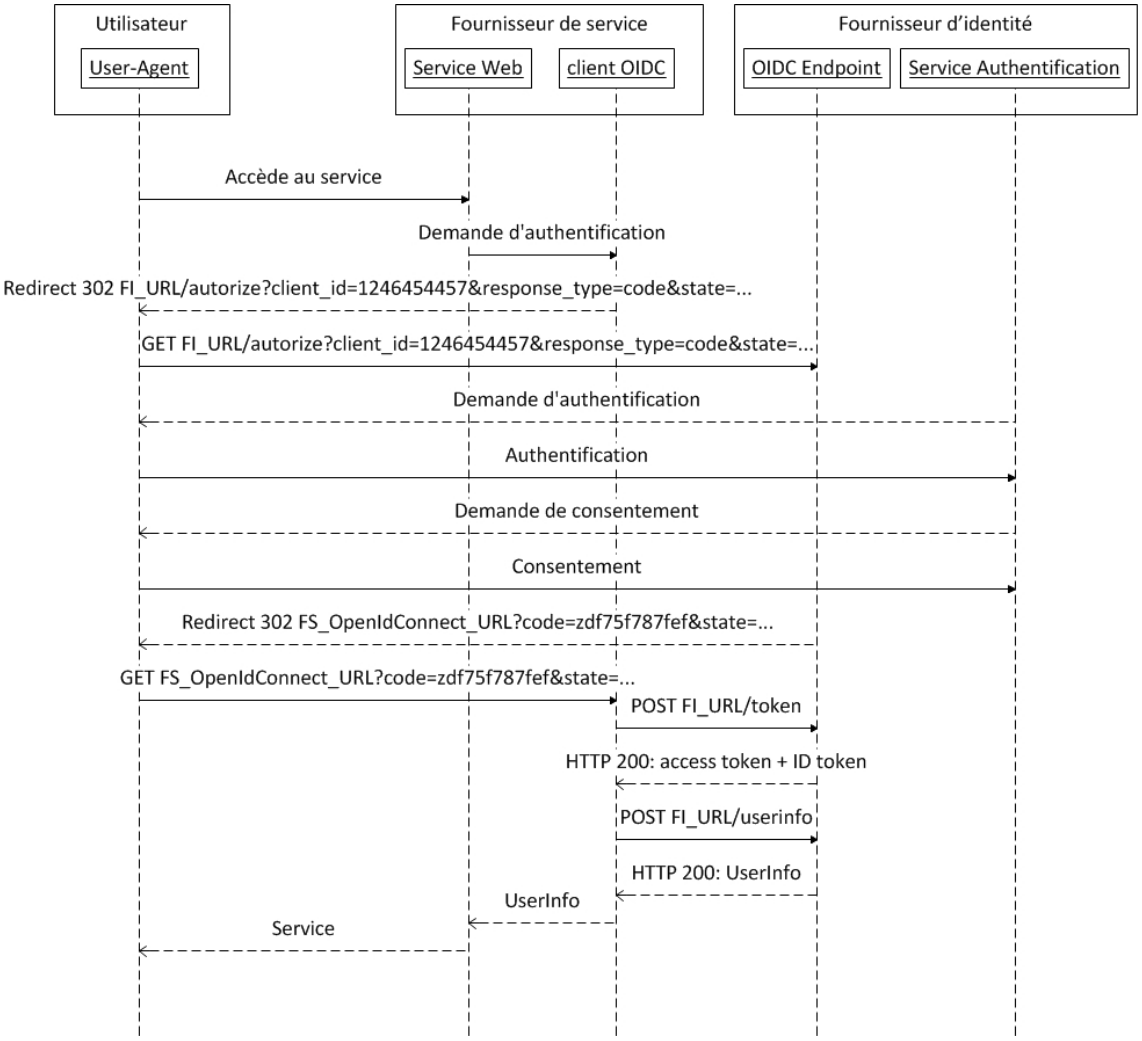


FIGURE 3.1 – Schéma simplifié des échanges de la cinématique *authorization code*

Attention

Dans le schéma 3.1, les services *Autorization Endpoint*, *Token Endpoint* et *UserInfo Endpoint* sont confondus en un seul service nommé *OIDC Endpoint*.

1. L'utilisateur souhaite accéder à une ressource protégée ;
2. Le fournisseur de service effectue une demande d'authentification par redirection. Les principaux paramètres [1] sont les suivants :
   - *client_id* : l'identifiant du client OIDC enregistré dans le système du fournisseur d'identité,
   - *response_type* : détermine la cinématique et les éléments envoyés dans la réponse de l'*OIDC Endpoint*. Dans le cadre de cette cinématique, sa valeur est « code »,
   - *scope* : ce paramètre détermine les *claims* demandés par le fournisseur de service. Par exemple, il peut contenir les valeurs « *openid* » et « *profile* » pour demander des informations définies par la norme OIDC sur le profil de l'utilisateur. Les *claims* peuvent également être demandés par le paramètre *claims* ou directement dans un JWT. La liste possible des *scopes* est définie par la norme OIDC[1],
   - *redirect_URI* : l'URL de redirection vers le client OIDC du fournisseur de service,
   - *state* : ce paramètre contient une valeur aléatoire liée à la session de l'utilisateur, il est renvoyé suite à l'authentification de l'utilisateur. Il permet de contrer les attaques de type CSRF,
   - *nonce* : ce paramètre contient une valeur aléatoire unique, il est renvoyé dans les paramètres de l'*ID Token*. Il permet de contrer les attaques de type rejeu ;
3. l'*OIDC Endpoint* (ou *Autorization Endpoint*) demande à l'utilisateur de s'authentifier ;
4. l'utilisateur s'authentifie auprès du fournisseur d'identité ;
5. l'*OIDC Endpoint* (ou *Autorization Endpoint*) demande le consentement à l'utilisateur de fournir les informations demandées selon le *scope* ou *claims* ;
6. l'utilisateur donne son consentement ;
7. l'*OIDC Endpoint* (ou *Autorization Endpoint*) redirige l'utilisateur vers le client OIDC grâce à la *redirect_URI* et en incluant les paramètres suivants :
   - *code* : ce paramètre correspond à l'*authorization code* généré par l'*OIDC Endpoint* (ou *Autorization Endpoint*),
   - *state* : ce paramètre correspond à celui envoyé dans la requête de demande d'authentification ;
8. le client OIDC du fournisseur de service effectue une requête serveur à serveur vers l'*OIDC Endpoint* (ou le *Token Endpoint*) avec les paramètres suivants :
   - *code* : ce paramètre correspond à l'*authorization code* précédemment retourné par l'*OIDC Endpoint* (ou *Autorization Endpoint*),
   - *grant_type* : ce paramètre indique à l'*OIDC Endpoint* (ou *Token Endpoint*) le type de cinématique, soit la valeur « *authorization_code* » dans le cas décrit ;
9. le serveur répond sous forme d'un JSON comprenant les éléments suivants :
   - *access_token* : contient l'*access token* qui permet au client OIDC de récupérer les informations consenties par l'utilisateur suite à son authentification,
   - *token_type* : précise le type de *token* reçu et les modalités pour récupérer les ressources associées. La norme OIDC utilise uniquement le type « Bearer » [11]. Il autorise toute personne ou entité possédant un *access token* de ce type d'accéder aux ressources associées sans s'authentifier auprès du *Resources Endpoint*,

1. La liste exhaustive des paramètres est décrite dans la documentation de la norme OIDC[1].

- *expires_in* : durée de validité de l'*access token*,
- *id_token* : contient l'*ID token* défini dans le paragraphe 2.2 ;

10. le client OIDC effectue une requête serveur à serveur vers le *UserInfo Endpoint* en mettant l'*access token* précédemment reçu dans l'entête *Authorization* ;
11. le serveur d'identité vérifie la validité de l'*access token* reçu puis fournit dans la réponse le *UserInfo* contenant les *claims* associés au scope.

3.1 Les communications

OIDC repose sur le protocole HTTP et sur l'utilisation de TLS pour sécuriser les échanges entre les différents acteurs indiqués dans le schéma 2.1.

On peut distinguer les communications HTTP :

- entre l'utilisateur et le site Web du fournisseur de service ;
- entre l'utilisateur et l'*Authorization Endpoint* ;
- entre le client OIDC et le *Token Endpoint* ;
- entre le client OIDC et le *UserInfo Endpoint*.

Le chiffrement des communications à l'aide du protocole TLS permet de protéger les communications en confidentialité et en intégrité.

R3

Mettre en œuvre HTTPS

Toutes les communications HTTP dans le cadre de la norme OIDC ainsi que celles entre l'utilisateur et le site du fournisseur de service doivent mettre en œuvre le protocole TLS.

R4

Appliquer les recommandations de sécurité relatives à TLS

TLS est l'un des principaux mécanismes de sécurité d'OIDC. Il est recommandé de mettre en place les bonnes pratiques décrites dans le guide TLS[5] publié par l'ANSSI.

L'utilisation d'un certificat serveur permet à l'utilisateur d'authentifier les serveurs du fournisseur de service :

- lors de la demande d'accès à un service ou une ressource protégé ;
- lors de la redirection de la réponse à la demande d'authentification.

Il permet également au fournisseur de service d'authentifier les serveurs du fournisseur d'identité

- lors des appels pour récupérer l'*ID token* et l'*access token* ;
- lors des appels pour récupérer les *claims UserInfo*.

Les communications serveur à serveur entre le client OIDC et les différents points de terminaison du fournisseur d'identité sont maîtrisées par chaque acteur, elles peuvent faire l'objet de mesures de sécurité supplémentaires.

R5

Imposer aux clients OIDC des suites cryptographiques recommandées pour les communications serveur à serveur

Il est recommandé au fournisseur d'identité d'imposer aux fournisseurs de service d'utiliser des suites cryptographiques robustes fournies dans le guide TLS[5] publié par l'ANSSI.

Des exemples de suites cryptographiques recommandées sont fournis dans une annexe du guide TLS[5].

Les fournisseurs d'identités sont des entités répertoriables et dont les URLs et les certificats ne varient pas fréquemment. Le fournisseur de service peut vérifier la cohérence du nom de domaine par rapport au certificat serveur et utiliser le *certificate pinning* pour s'assurer de l'identité des serveurs contactés.

R6

Vérifier le nom de domaine

Il est recommandé au fournisseur de service de vérifier la correspondance entre le nom de domaine du fournisseur d'identité et les valeurs possibles dans le *SubjectAltName* du certificat présenté lors de la connexion TLS.

R7

Utiliser le certificate pinning

Il est recommandé au fournisseur de service de mettre en œuvre le mécanisme *certificat pinning*. Il consiste à précharger dans le client OIDC :

- soit le certificat serveur ;
- soit son empreinte ;
- soit sa clé publique pour être comparée avec celle reçue lors des échanges TLS.

Attention

L'implémentation de la recommandation R7 implique une mise à jour du certificat préchargé en cas de renouvellement du certificat serveur.

3.2 La demande d'authentification

La demande d'authentification est générée par le fournisseur de service lorsqu'un utilisateur accède à une ressource exigeant le contrôle de son identité ou lorsque l'utilisation du service nécessite de demander l'autorisation d'accéder à des données possédées par un fournisseur de données.

Les paramètres envoyés dans la requête HTTP de la demande d'authentification à l'étape 2 du schéma 3.1 peuvent être transmis :

- dans les paramètres de l'URL d'une requête HTTP GET de redirection ;
- dans le corps d'une requête HTTP POST à l'aide par exemple de JavaScript ;
- sous forme d'un JWT (ou JWS) transmis par le paramètre *request* d'une requête HTTP GET ou POST ;
- par une URL, transmise par le paramètre *request_uri* d'une requête HTTP GET ou POST, qui référence un JWT (ou JWS). Celui-ci est récupéré en accédant à cette URL.

Un JWT dans une demande d'authentification est également un moyen de demander la transmission de *claims* supplémentaires ne faisant pas partie de ceux identifiables par le paramètre *scope*. Les paramètres transmis sont alors accessibles par l'utilisateur du service et donc manipulables. L'utilisation d'un JWS garantit leur intégrité et empêche ainsi leur manipulation.

La vérification du HMAC se fait grâce au secret partagé nommé *client_secret* entre le fournisseur de service et le fournisseur d'identité lors de l'enrôlement du client OIDC.

R8

Utiliser un JWS protégé par un HMAC

Pour éviter la manipulation de paramètres sensibles dans une demande d'authentification, il est recommandé qu'un client OIDC envoie les paramètres dans un JWS protégé par un HMAC.

La signature apporte la propriété de non répudiation par rapport à un HMAC, elle est générée grâce à un algorithme cryptographique asymétrique et une clé privée que seul le fournisseur de service possède. La vérification de cette signature grâce à une clé publique garantit l'origine de la demande en plus de son intégrité.

R8 +

Utiliser un JWS protégé par une signature

Pour éviter la manipulation de paramètres sensibles d'une demande d'authentification et garantir son origine, il est recommandé qu'un client OIDC envoie tous les paramètres dans un JWS protégé par une signature électronique.

Attention

La signature n'est valide que si la clé publique utilisée pour la vérifier a été transmise de manière intègre, par exemple par l'utilisation d'un certificat lui même signé par une autorité reconnue.

La norme OIDC n'impose pas pour un client OIDC un seul mode de transmission des paramètres pour une demande d'authentification au fournisseur d'identité. Ce choix est laissé au fournisseur de service. Un attaquant peut tenter de manipuler les paramètres et donc de forger des demandes valides sans utiliser le paramètre *request* ou en utilisant un JWS sans garantie d'intégrité grâce à la valeur « *none* » dans le paramètre *alg*.

R9

Imposer un mode de transmission des paramètres dans une demande d'authentification

Il est recommandé au fournisseur d'identité de n'autoriser qu'un seul mode de demande d'authentification par client OIDC. Par exemple, un client OIDC ayant opté pour un JWS signé ne doit pas pouvoir transmettre autrement les paramètres dans l'URL. Le mode de transmission et le mécanisme de protection (signature ou HMAC) doivent être fixés lors de l'enrôlement du client par le fournisseur de service.

Attention

Une implémentation certifiée par la fondation OpenID respecte strictement la norme OIDC, celle-ci n'indique pas de restreindre au client OIDC un unique mode de transmission des paramètres (JWS, chaines de paramètre etc). Une stricte conformité ne permettra donc pas la mise en œuvre de la recommandation R9 et diminue fortement l'intérêt de mettre en œuvre les recommandations R8 et R8+.

La norme OIDC recommande l'utilisation du paramètre *state* sans le rendre obligatoire dans une demande d'authentification. Ce paramètre permet de lier une demande d'authentification à sa réponse. Un fournisseur de service qui ne l'utilise pas est donc vulnérable à des attaques de type CSRF.

R10

Systématiser l'envoi du paramètre state

Pour se protéger contre les attaques CSRF, il est recommandé que les clients OIDC d'un fournisseur de service envoient systématiquement le paramètre *state* dans les demandes d'authentification.

Un attaquant ne doit pas pouvoir prédire les valeurs du paramètre *state* générées par le client OIDC, ce qui lui permettrait de contourner la protection contre les attaques CSRF.

R11

Générer aléatoirement le paramètre state

Le paramètre *state* doit contenir une valeur aléatoire générée à l'aide d'une fonction utilisant un générateur d'aléa cryptographique et ayant une entropie minimale de 128 bits. Cette taille peut être atteinte en générant aléatoirement une chaîne de 22 caractères ASCII imprimables (A à Z, a à z et 0 à 9).

Attention

Les implémentations certifiées par la fondation OpenID respectent strictement la norme OIDC en acceptant des demandes d'authentification sans le paramètre *state*

pour un client OIDC. Un fournisseur d'identité ne pourra pas imposer aux clients OIDC d'utiliser le paramètre *state*.

R12

Détecter les demandes d'authentification sans le paramètre state

Le fournisseur d'identité doit mettre en place un système de détection des demandes d'authentification ne contenant pas le paramètre *state* afin de prévenir au plus tôt les tentatives de fraude.

Un fournisseur de service peut avoir besoin de conserver des informations, ou le contexte de la session, d'un utilisateur. Pour cela, il a besoin de conserver l'état de la session afin de poursuivre l'action en cours après l'authentification. Stocker en base de données le contexte de la session et utiliser le paramètre *state* en tant qu'identifiant pour les retrouver n'est pas une bonne pratique. Elle expose les données de l'utilisateur par le simple rejeu de l'URL d'une réponse à une demande d'authentification. Il est préférable de récupérer grâce à un cookie de session le paramètre *state* et les données de session conservés en mémoire.

R13

Utiliser les cookies de session

Il est recommandé que le client OIDC conserve le paramètre *state* dans la session de l'utilisateur qui sera récupérée grâce à un cookie de session.

Le paramètre *nonce* est optionnel dans une demande d'authentification de la cinématique *authorization code*. Ce paramètre permet de se protéger contre les attaques par rejeu.

R14

Systématiser l'envoi du paramètre nonce

Pour se protéger contre les attaques par rejeu de requêtes, il est recommandé que les clients OIDC d'un fournisseur de service utilisent systématiquement le paramètre *nonce* dans les demandes d'authentification.

Un attaquant ne doit pas pouvoir prédire les valeurs du paramètre *nonce* générées par le client OIDC, ce qui lui permettrait de contourner la protection contre les attaques par rejeu.

R15

Générer aléatoirement le paramètre nonce

Le paramètre *nonce* doit contenir une valeur aléatoire générée à l'aide d'une fonction utilisant un générateur d'aléa cryptographique et ayant une entropie minimale de 128 bits. Cette taille peut être atteinte en générant aléatoirement une chaîne de 22 caractères ASCII imprimables (A à Z, a à z et 0 à 9).

Attention

Les implémentations certifiées par la fondation OpenID respectent strictement la norme OIDC en acceptant des demandes d'authentification sans le paramètre *nonce* pour un client OIDC. Un fournisseur d'identité ne pourra pas imposer aux clients OIDC d'utiliser le paramètre *nonce*.

R16

Détecter les demandes d'authentification sans le paramètre nonce

Le fournisseur d'identité doit mettre en place un système de détection des demandes d'authentification ne contenant pas le paramètre *nonce* afin de prévenir au plus tôt les tentatives de fraude.

Une demande d'authentification contient le paramètre *redirect_uri* dont la valeur est l'URL du client *OIDC* d'un fournisseur de service. Cette URL est utilisée par le fournisseur d'identité pour rediriger l'utilisateur après son authentification. Sans vérification de cette URL, un attaquant pourrait par exemple forger une demande contenant une URL vers un site malveillant et hameçonner un utilisateur pour voler l'*Authorization Code*.

R17

Se protéger contre les redirections ouvertes

Pour se protéger contre les redirections ouvertes, le fournisseur d'identité doit vérifier l'appartenance de l'URL reçue dans le paramètre *redirect_uri* à une liste blanche d'URL associée à l'identifiant du client OIDC. Cette liste aura été fournie lors de l'enrôlement. De plus, cette vérification peut avoir lieu dès la réception de la demande d'authentification.

3.3 La réponse à une demande d'authentification

Suite à l'authentification de l'utilisateur, l'*Authorization Endpoint* génère un *authorization code* qui permettra au client OIDC de récupérer ultérieurement un *access token* et un *ID Token*. Le paramètre *code* contenant l'*authorization code* est transmis au client OIDC dans l'URL du paramètre *redirect_uri* au travers du navigateur de l'utilisateur. Un attaquant ne doit pas pouvoir prédire des *authorization code*.

R18

Générer aléatoirement les authorization code

L'*authorization code* doit être généré à l'aide d'une fonction utilisant un générateur d'aléa cryptographique et atteindre une entropie minimale de 128 bits. Cette taille peut être atteinte en générant aléatoirement une chaîne de 22 caractères ASCII imprimables (A à Z, a à z et 0 à 9).

Un *authorization code* peut être récupéré dans les traces d'un serveur mandataire ou sur un poste utilisateur. Un attaquant pourrait le réutiliser pour usurper ultérieurement l'identité de l'utilisateur sur un service n'implémentant pas la recommandation R14.

R19

Limiter la durée de vie d'un authorization code

Le fournisseur d'identité doit limiter la durée de vie d'un *authorization code* au temps de récupération de l'*ID Token* et de l'*access token* qui lui sont associés. Quelques minutes suffisent généralement. Une fois cette durée dépassée, l'*authorization code* doit être désactivé ou supprimé.

R20

Associer le code authorization code au client OIDC

Pour limiter la réutilisation d'un *authorization code* sur des services différents, il est recommandé au fournisseur d'identité d'associer l'*authorization code* à l'identifiant du client OIDC (*client_id*) qui est à l'origine de la demande authentification.

Les *authorization code* sont généralement stockés en base de données et sont donc potentiellement récupérables par des failles applicatives de type injection SQL ou par un administrateur d'un fournisseur d'identité mal intentionné.

R21

Stocker les authorization code sous forme d'empreinte

Il est recommandé de stocker les *authorization code* sous forme d'empreinte en utilisant une fonction de hachage tel que SHA256 qui est compatible avec le RGS [6].

Si le client OIDC implémente la recommandation R10, le paramètre *state* sera présent dans l'URL de redirection *redirect_uri* en plus de l'*authorization code*.

R22

Vérifier le paramètre state associé à la session de l'utilisateur

Le client OIDC doit vérifier la correspondance entre la valeur du paramètre *state* reçue dans l'URL de redirection et celle stockée dans la session de l'utilisateur (voir recommandation R13).

3.4 L'access token et l'ID Token

Pour récupérer l'*access token* et l'*ID Token*, le client OIDC doit s'authentifier auprès du *Token Endpoint*. Par niveau de sécurité croissant, il peut le faire en utilisant :

- un mot de passe envoyé par une authentification *HTTP Basic*, ou dans le corps d'une requête HTTP POST ;
- un JWS avec un HMAC ;
- un JWS avec une signature.

R23

Utiliser une authentification du client OIDC adaptée

Pour récupérer l'*access token*, il est recommandé d'utiliser une authentification du client OIDC adaptée au niveau de sensibilité des informations récupérables et déterminée par une analyse de risque.

Une fois le client OIDC authentifié, le client OIDC échange l'*authorization code* contre un *access token* et un *ID Token* définis dans le paragraphe 2.2. La norme OIDC ne définit pas la représentation d'un *access token*. Une implémentation possible des *access token* citée par la norme est la génération d'un aléa. Il est alors associée par le *Token Enpoint* aux *Claims* ou *UserInfo* définis par le paramètre *scope*. Une authentification n'étant pas nécessaire pour les récupérer, un attaquant ne doit donc pas être en capacité de forger des *access token* valides. D'autres solutions sont bien sûr possibles, comme par exemple utiliser l'*ID Token* en tant qu'*access token*.

R24

Générer aléatoirement les access token

Les *access token* sous forme d'aléa doivent être générés à l'aide d'une fonction utilisant un générateur d'aléa cryptographique et avoir une entropie minimale de 128 bits. Cette taille peut être atteinte en générant aléatoirement une chaîne de 22 caractères ASCII imprimables (A à Z, a à z et 0 à 9).

En fonction de la durée de vie des *access token*, les fournisseurs d'identité peuvent être amenés à les stocker en base de données. Ceci entraîne des risques d'accès non autorisé que cela soit par une faille de type injection SQL ou un administrateur du fournisseur d'identité.

R25

Stocker les access token sous formes d'empreintes

Si les *access token* sont stockés en base de données, il est recommandé de les conserver sous forme d'une empreinte en utilisant une fonction de hachage telle que SHA256 qui est compatible avec le RGS[6]

OIDC a ajouté au protocole OAuth2.0 la notion d'*ID Token*. Il contient des informations sur l'authentification de l'utilisateur telles que :

- *iss* : l'URL du serveur du fournisseur d'identité qui a généré l'*ID Token* ;
- *aud* : l'identifiant du client OIDC pour lequel l'*ID Token* a été généré ;
- *sub* : un identifiant unique de l'utilisateur ;

- *iat* : date à laquelle l'*ID Token* a été généré ;
- *exp* : date à laquelle l'*ID Token* expire ;
- *nonce* (optionnelle) : valeur aléatoire transmise dans la demande d'authentification.

Il est transmis sous la forme d'un JWS protégé en intégrité par un HMAC ou une signature électronique. Dans le cadre de la cinématique *authorization code*, l'ID token est protégé en intégrité par le flux TLS entre le client OIDC et le *Token Endpoint*. Cependant, vérifier l'intégrité de l'*ID Token* est une bonne pratique.

R26

Vérifier l'intégrité de l'ID Token

Dans le cadre d'une mesure de défense en profondeur, le client OIDC peut vérifier le HMAC ou la signature électronique de l'*ID Token* lors de sa réception.

Pour générer le HMAC de l'*ID Token*, la norme OIDC indique d'utiliser le *client_secret*. Ce secret peut également être utilisé par le *Token Endpoint* en tant que mot de passe pour authentifier le client OIDC. Or, cette double utilisation empêche de stocker le *client_secret* dans les règles de l'art en utilisant une fonction de hachage et un sel.

R27

Utiliser un secret partagé différent du client_secret pour générer le HMAC de l'ID Token

Dans le cas d'une authentification par mot de passe d'un client OIDC, il est recommandé d'utiliser un secret partagé différent du *client_secret* pour générer les HMAC de l'*ID Token*.

Attention

Une implémentation certifiée par la fondation OpenID respecte la norme OIDC. Cette conformité ne permettra donc pas la mise en œuvre de la recommandation R27.

Dans certains cas, l'*id_token* est utilisé en tant que jeton de session, ou est identique à l'*acces token*, il peut être opportun de le chiffrer afin d'éviter la divulgation d'information.

Le client OIDC peut utiliser les informations reçues dans l'*ID Token* pour vérifier qu'il n'a pas été rejoué ou qu'il lui est bien destiné. On peut citer par exemple :

- comparer la valeur du champ *nonce* avec celle envoyée dans la demande d'authentification et conservée dans la session de l'utilisateur ;
- comparer la valeur du champ *iss* avec l'URL de l'*Authorization Endpoint* utilisée dans la demande d'authentification ;
- comparer la valeur du champ *aud* avec son identifiant client (*client_id*) ;
- vérifier que l'*ID Token* est valide grâce aux champs *iat* et *exp*.

R28

Vérifier les informations d'un ID Token

Le client OIDC doit faire les vérifications qui concernent l'*ID Token* indiquées dans les spécifications de la norme OIDC[1].

L'*ID Token* peut contenir des informations supplémentaires sur l'authentification grâce au champ *acr*, ces informations sont transmises lors de la demande d'authentification en utilisant le paramètre *acr_values*. Un fournisseur d'identité peut proposer plusieurs moyens d'authentification ayant des niveaux de sécurité différents (identifiant/mot de passe associé à un code à usage unique, une carte à puce protégée par un code, etc.). Un fournisseur de service peut demander le respect du niveau d'authentification demandé pour accéder au service en comparant le champ *acr* de l'*ID Token* avec la valeur transmise dans la demande d'authentification.

R29

Vérifier le niveau d'authentification de l'utilisateur

Un fournisseur de service ayant demandé un niveau d'authentification minimal de l'utilisateur en précisant une valeur dans le paramètre *acr_values* doit comparer sa valeur avec le champ *acr* contenu dans l'*ID Token*.

Une mesure de défense en profondeur consiste à ne pouvoir utiliser l'*authorization code* qu'une seule fois.

R30

Rendre inutilisables les authorization code

Une fois l'*access token* et l'*ID Token* délivrés au client OIDC, il est recommandé au fournisseur d'identité de désactiver ou de supprimer l'*authorization code*.

3.5 Récupération du UserInfo

Le protocole OAuth2.0 définit la notion de ressources appartenant à un utilisateur. OIDC utilise cette notion pour définir les *Claims* pouvant représenter son identité. Voici quelques exemples :

- *sub* : identifiant unique ;
- *name* : nom complet pour affichage ;
- *given_name* : prénom ;
- *last_name* : nom de famille ;
- *birth_day* : date d'anniversaire ;
- *email* : adresse email ;
- *address* : adresse postale.

Le fournisseur de service peut récupèrer les *Claims* définis par le paramètre *scope* et consentis par l'utilisateur. Pour cela, le client OIDC transmet au *UserInfo Endpoint* l'*access token* précédemment récupéré. Ce dernier permet donc d'accéder à des informations personnelles sans authentification et doit être considéré comme sensible.

R31

Transmettre les access token par l'entête Authorization

L'*access token* doit être envoyé en utilisant l'entête *Authorization* comme défini dans la norme OIDC car les serveurs mandataires inverses et les serveurs Web n'écrivent pas cet entête dans les journaux.

R32

Ne pas écrire dans les journaux les access token

L'*access token* est une information sensible, il ne doit jamais être écrit dans les journaux des évènements des applications qui le manipulent.

Un *access token* volé peut être utilisé par n'importe quel attaquant, la norme prévoit une durée de vie pour diminuer le risque d'utilisation frauduleuse.

R33

Restreindre la durée de validité d'un access token

L'*access token* doit avoir une durée de validité la plus courte possible. Il convient d'effectuer une analyse de risque afin de déterminer cette durée en fonction du niveau de risque et des besoins.

R34

Croiser les informations du UserInfo et de l'ID Token

Il est recommandé au client OIDC de vérifier que l'identifiant unique de l'utilisateur *sub* reçu dans le *UserInfo* est identique à celui contenu dans l'*ID Token*.

3.6 Gestion des secrets d'authentification

L'authentification par mot de passe ou par un HMAC nécessite le partage d'un secret entre le fournisseur de service et le fournisseur d'identité. Ce secret est nommé *client_secret* par la norme OIDC.

R35

Générer aléatoirement les secrets d'authentification partagés

Les secrets d'authentification partagés (mot de passe, clé pour générer les HMAC) doivent être générés à l'aide d'une fonction utilisant un générateur d'aléa cryptographique et avoir une entropie minimale de 128 bits. Cette taille peut être atteinte en générant aléatoirement une chaîne de 22 caractères ASCII imprimables (A à Z, a à z et 0 à 9).

Le vol d'un secret peut être silencieux, son renouvellement permet de repérer et de stopper les éventuelles usurpations d'un fournisseur de service.

R36

Renouveler les secrets d'authentification partagés

Les secrets d'authentification partagés (mot de passe, clé pour générer les HMAC) utilisés par les clients OIDC doivent être renouvelés régulièrement selon la politique de sécurité du système.

La compromission d'un secret d'un client OIDC ne doit pas compromettre les autres clients OIDC.

R37

Utiliser un secret différent par client OIDC

Il est recommandé d'utiliser un secret d'authentification différent par client OIDC.

R38

Restreindre l'accès au secret d'authentification

Il est recommandé de restreindre et de contrôler strictement l'accès au secret d'authentification.

L'utilisation d'un HMAC dans le JWS a l'avantage de ne pas envoyer le secret dans les échanges et de permettre la vérification de l'intégrité des paramètres.

De même, une signature dans un JWS permet de vérifier l'intégrité des paramètres et d'authentifier un client OIDC par l'utilisation d'une clé privée et d'une clé publique. L'authentification se fait soit par :

- un certificat ;
- la récupération d'une clé publique par interrogation d'une URL, en utilisant le protocole HTTPS. L'authentification et l'intégrité sont réalisées dans ce cas grâce au contrôle du nom de domaine, au protocole TLS et au certificat serveur associé.

Attention

La signature n'est valide que si la clé publique utilisée pour la vérifier a été transmise de manière intègre, par exemple par l'utilisation d'un certificat lui même signé par une autorité reconnue.

R39

Utiliser des certificats pour authentifier les JWS

Il est recommandé d'utiliser des certificats signés par une autorité de confiance pour vérifier la signature d'un JWS.

R40

Restreindre l'accès à la clé privée de signature

Il est recommandé de restreindre et de contrôler strictement l'accès à la clé privée utilisée pour signer les JWS.

R41

Vérifier la révocation des certificats

Il est recommandé au fournisseur d'identité de mettre en œuvre un mécanisme vérifiant la révocation des certificats utilisés pour signer les JWS.

3.7 Les JSON Web Signature

OIDC s'appuie sur la RFC 7515 [9] pour la génération d'un JWS, sur la RFC 7516 [12] pour la génération d'un JWE et la RFC 7518 [8] pour la liste des algorithmes cryptographiques (JWA). Les

spécifications OIDC ne font pas de distinction du point de la sécurité entre le mécanisme d'authentification réalisé par une signature et celui réalisé par un HMAC, bien que ce dernier n'apporte pas le même niveau de garantie. En effet la clé privée pour générer la signature n'est pas partagée contrairement à la clé utilisée pour générer le HMAC.

R42

Utiliser des fonctions de hachage recommandées

La fonction de hachage employée pour générer le HMAC d'un JWS doit être compatible avec les recommandations de sécurité décrites dans les annexes B1 et B2 du RGS [6]. Il est recommandé d'utiliser au minimum la fonction de hachage SHA256.

R43

Utiliser des mécanismes de signature recommandés

La fonction de signature des JWS doit utiliser un mécanisme conforme aux recommandations de sécurité décrites dans les annexes B1 et B2 du RGS [6].

Attention

RSASSA-PKCS1-V1.5, qui fait partie des algorithmes indiqués dans la RFC 7518 [8], n'est pas conforme au RGS lorsque certaines conditions ne sont pas respectés (voir annexe B1 [6]).

La RFC 7515 [9] définit le type d'algorithme utilisé pour générer le HMAC ou la signature du JWS par le champ *alg*. Elle permet donc à un attaquant de choisir l'algorithme de vérification de signature. De nombreuses implémentations de cette RFC ont fait l'objet d'une faille de sécurité permettant de forger des JWS avec un HMAC valide à partir d'une clé publique. De plus, la RFC prévoit la possibilité de générer des JWS dont l'intégrité n'est pas garantie en précisant la valeur « *none* » pour le paramètre *alg*.

R44

Fixer l'algorithme utilisé pour le JWS

Il est recommandé de ne pas utiliser le champ *alg* du JWS. L'algorithme, utilisé pour la génération et la vérification du HMAC ou de la signature, doit être fixé lors de l'enrôlement du client OIDC.

3.8 L'application Web OIDC

Un service OIDC mise en œuvre par le fournisseur d'identité est une application Web sensible exposée sur Internet, elle centralise les authentifications et la gestion des droits d'accès aux informations des utilisateurs. Il en est de même pour le client OIDC. Ces applications sont vulnérables aux failles classiques du *Top Ten* OWASP.

R45

Sécuriser l'application Web OIDC

Il est recommandé d'appliquer les recommandations du guide sur la sécurité des applications Web [3] de l'ANSSI.

De manière non exhaustive, les fournisseurs de service et d'identité doivent :

- valider tous les paramètres entrants sous le contrôle d'un attaquant. Par exemple, le *client_id* ne doit contenir que des chiffres ou le paramètre *response_type* ne doit contenir que des valeurs autorisées (*code*, *token access token* etc) ;
- utiliser des requêtes paramétrées dans les requêtes SQL pour empêcher les injections SQL ;
- durcir les différents composants installés (base de données, serveur d'application, serveur mandataire inversé, ….) ;
- mettre en place une architecture n-tiers pour mettre en œuvre des mécanismes de protection indépendants dans le cadre de la défense en profondeur. Par exemple, la mise en place d'un serveurs mandataire inversé en amont rend possible un filtrage réseau et applicatif en plus de celui présent dans l'application responsable des traitements.

Les journaux sont une source d'information riche. Celle-ci peut être utilisée a priori pour détecter des incidents de sécurité et a posteriori pour retrouver les traces d'un incident de sécurité.

R46

Mettre en oeuvre un système de journalisation

Il est recommandé d'appliquer les recommandations de sécurité du guide sur la mise en oeuvre d'un système de journalisation [4] de l'ANSSI.

R47

Journaliser les évènements importants

Il est recommandé de journaliser les évènements importants en s'appuyant sur l'annexe A du guide sur la mise en oeuvre d'un système de journalisation [4] de l'ANSSI. Une attention particulère doit être portée aux évènements liés à l'authentification.

4
Enrôlement des clients OIDC

Pour mettre en œuvre le service d'authentification, les fournisseurs de service et d'identité doivent s'échanger des informations (par exemple, le client_id, les secrets d'authentification, les *redirect_uri*, les URLs d'accès au *Authorization Endpoint*, au Token Endpoint, ….). Cet échange s'effectue lors de l'enrôlement des clients OIDC.

La norme OIDC spécifie un mécanisme pour automatiser la découverte de la configuration de fournisseurs d'identité et un pour l'enrôlement des clients OIDC. La combinaison de ces deux mécanismes facilite l'accès au service d'authentification, mais aussi la mise en œuvre d'une attaque nommée *IdP Mix-Up Attack*. Celle-ci est décrite sur le site OpenID[2]. Elle consiste à usurper l'identité d'un fournisseur d'identité auprès d'un fournisseur de service.

R48

Désactiver la découverte automatisée

Il est recommandé de ne pas utiliser la découverte automatisée de configuration d'un fournisseur d'identité pour minimiser les risques d'attaque de type *IdP Mix-Up Attack*.

R49

Ne pas utiliser l'enrôlement automatisé

Il est recommandé au fournisseur d'identité de ne pas mettre en œuvre l'automatisation de l'enrôlement des clients OIDC pour minimiser les risques d'attaque de type *IdP Mix-Up Attack*.

Le fournisseur d'identité peut fournir aux fournisseurs de service une interface Web permettant d'échanger les informations nécessaires à la mise en œuvre du service. Néanmoins, cette interface, qui correspond généralement à une application Web, est un vecteur d'attaque du système d'authentification. Celle-ci doit donc être également sécurisée.

R50

Sécuriser l'interface Web de configuration d'un client OIDC

L'interface de configuration des clients OIDC fournie par un fournisseur d'identité aux fournisseurs de service doit respecter les bonnes pratiques de sécurité (authentification, gestion des droits d'accès, validation des paramètres, etc).

En fonction du niveau de sensibilité des données accessibles, les fournisseurs de service peuvent être considérés comme malveillants, leur identité doit donc être vérifiée.

R51

Vérifier l'identité des fournisseurs de service

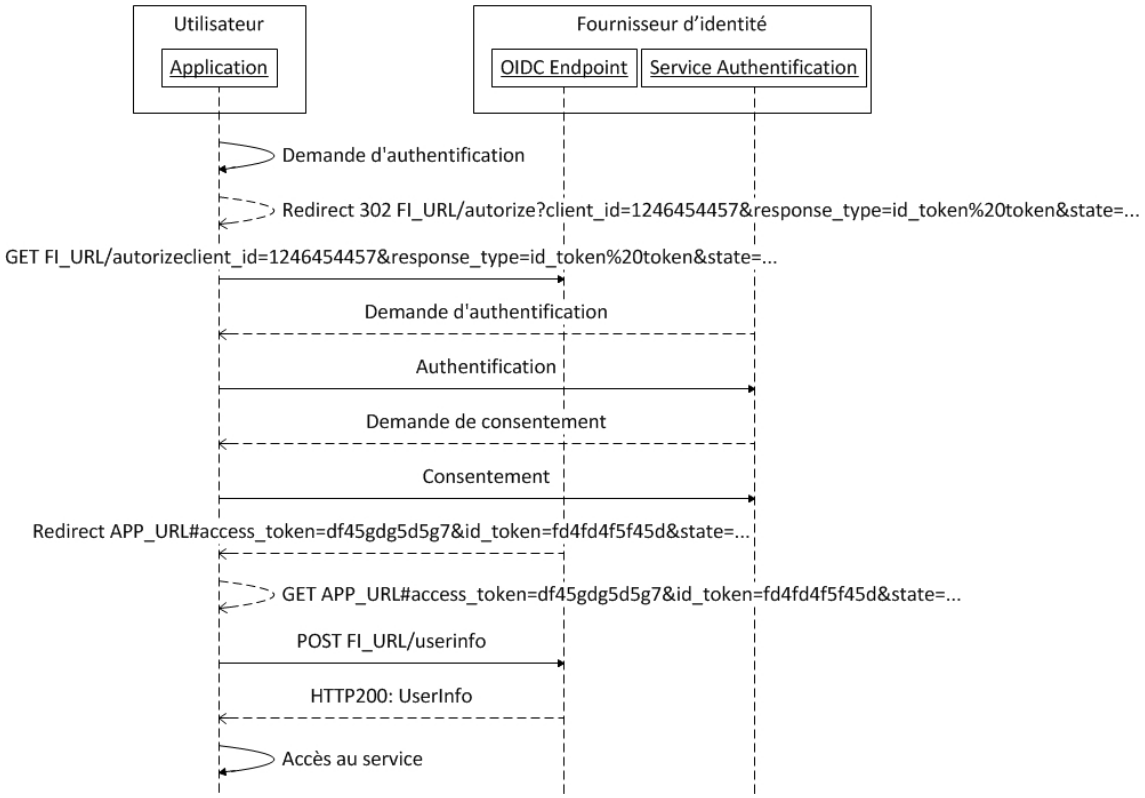
Afin de limiter l'utilisation du système aux seuls services légitimes, le fournisseur d'identité doit mettre en place une procédure d'enrôlement comprenant la vérification de l'identité des fournisseurs de service. Le niveau de vérification doit être adapté au niveau de sensibilité des informations accessibles par ce service.

Annexe A
La cinématique implicite

Cette annexe décrit les échanges effectués lors de la cinématique implicte.

Attention

Dans le schéma, les services *Token Endpoint* et *UserInfo Endpoint* sont confondus en un seul service nommé *OIDC Endpoint*.



1. L'utilisateur souhaite accéder à une ressource protégée ;
2. Le fournisseur de service effectue une demande d'authentification par redirection. Les principaux paramètres [2] sont les suivants :
   - *client_id* : l'identifiant du client OIDC enregistré dans le système du fournisseur d'identité,
   - *response_type* : détermine la cinématique et les éléments envoyés dans la réponse de l'*OIDC Endpoint*. Dans le cadre de cette cinématique, la valeur est « *id_token token* »,
   - *scope* : ce paramètre détermine les *claims* demandés par le fournisseur de service. Par exemple, il peut contenir les valeurs « *openid* » et « *profile* » pour demander des informations

2. La liste exhaustive des paramètres est décrite dans la documentation de la norme OIDC[1].

définies par la norme OIDC sur le profil de l'utilisateur. Les *claims* peuvent également être demandés par le paramètre *claims* ou directement dans un JWT. La liste possible des *scopes* est définie par la norme,

- *redirect_URI* : l'URL de redirection vers le client OIDC du fournisseur de service,
- *state* : ce paramètre contient une valeur aléatoire liée à la session de l'utilisateur, il est renvoyé suite à l'authentification de l'utilisateur. Il permet de contrer les attaques de type CSRF,
- nonce : ce paramètre contient une valeur aléatoire unique, il est renvoyé dans les paramètres de l'*ID Token*. Il permet de contrer les attaques de type rejeu ;

3. l'*OIDC Endpoint* (ou *Autorization Endpoint*) demande à l'utilisateur de s'authentifier ;
4. l'utilisateur s'authentifie auprès du fournisseur d'identité ;
5. l'*OIDC Endpoint* (ou *Autorization Endpoint*) demande le consentement à l'utilisateur de fournir les informations demandées selon le *scope* ou *claims* ;
6. l'utilisateur donne son consentement ;
7. l'*OIDC Endpoint* (ou *Autorization Endpoint*) redirige l'utilisateur vers le client OIDC grâce à la *redirect_URI* et en incluant les paramètres suivants :
   - *access_token* : ce paramètre est fourni par l'*OIDC Endpoint* (ou *Autorization Endpoint*) si le paramètre *[*response_type contient la valeur « *token* », il correspond à l'*access token* permettant de récupérer l'identité de l'utilisateur auprès du *UserInfo Endpoint*,
   - *id_token* : ce paramètre est fourni par l'*OIDC Endpoint* (ou *Autorization Endpoint*) si le paramètre *reponse_type* contient la valeur *id_token*,
   - *state* : ce paramètre correspond à celui envoyé dans la requête de demande d'authentification ;
8. le client OIDC effectue une requête serveur à serveur vers l'*OIDC Endpoint* (ou le *UserInfo Endpoint*) en mettant l'*access token* précédemment reçu dans l'entête *Authorization* ;
9. le serveur d'identité vérifie la validité de l'*access token* reçu puis fournit dans la réponse le *UserInfo* contenant les *claims* associés au scope.

Annexe B
Quelle entité doit implémenter quelles recommandations ?

Le but de cette annexe est d'indiquer les recommandations devant être mises en œuvre soit par les fournisseurs de service, soit par les fournisseurs d'identité, soit par les deux.

Fournisseur de service	Fournisseur d'identité
R3 R4 R6 R7 R8 R8+ R10 R11 R13 R14 R15 R22 R26 R27 R28 R31 R32 R34 R40 R42 R44 R45 R46 R47	R1 R2 R3 R4 R5 R9 R12 R16 R17 R18 R19 R20 R21 R23 R24 R25 R29 R30 R32 R33 R35 R36 R37 R38 R39 R41 R42 R43 R44 R45 R46 R47 R48 R49 R50 R51

Liste des recommandations

Bibliographie

[1] *OpenID Connect (OIDC)*.
OpenID Connect Core Specification v1.0, OpenID Foundation, nov 2014.
http://openid.net/specs/openid-connect-core-1_0.html.

[2] *OpenID Connect Idp Mix-Up Attack*.
OpenID Connect Attack v1.0, OpenID Foundation, jul 2016.
https://openid.net/2016/07/16/preventing-mix-up-attacks-with-openid-connect/.

[3] *Recommandations relatives pour la sécurisation des sites web*.
Note technique DAT-NT-009/ANSSI/SDE/NP v1.1, ANSSI, août 2013.
https://www.ssi.gouv.fr/securisation-sites-web.

[4] *Recommandations de sécurité pour la mise en œuvre d'un système de journalisation*.
Note technique DAT-NT-012/ANSSI/SDE/NP v1.0, ANSSI, décembre 2013.
https://www.ssi.gouv.fr/journalisation.

[5] *Guide TLS*.
Guide SDE-NT-035 v1.1, ANSSI, août 2016.
https://www.ssi.gouv.fr/nt-tls.

[6] *RGS : Référentiel Général de Sécurité*.
Référentiel Version 2.0, ANSSI, juin 2012.
https://www.ssi.gouv.fr/rgs.

[7] *The OAuth 2.0 Authorization Framework*.
D. Hardt.
RFC 6749, RFC Editor, oct 2012.
http://www.rfc-editor.org/rfc/rfc6749.txt.

[8] *JSON Web Algorithms (JWA)*.
M. Jones.
RFC 7518, RFC Editor, mai 2015.
http://www.rfc-editor.org/rfc/rfc7518.txt.

[9] *JSON Web Signature (JWS)*.
M. Jones, J. Bradley, and N. Sakimura.
RFC 7515, RFC Editor, mai 2015.
http://www.rfc-editor.org/rfc/rfc7515.txt.

[10] *JSON Web Token (JWT)*.
M. Jones, J. Bradley, and N. Sakimura.
RFC 7519, RFC Editor, mai 2015.
http://www.rfc-editor.org/rfc/rfc7519.txt.

[11] *The OAuth 2.0 Authorization Framework : Bearer Token Usage*.
M. Jones and D. Hardt.
RFC 6750, RFC Editor, oct 2012.
http://www.rfc-editor.org/rfc/rfc6750.txt.

[12] *JSON Web Encryption (JWE)*.
M. Jones and J. Hildebrand.
RFC 7516, RFC Editor, mai 2015.
http://www.rfc-editor.org/rfc/rfc7516.txt.

ANSSI-PA-080
Version 1.0 - 08/09/2020

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de La Tour-Maubourg, 75700 PARIS 07 SP
www.ssi.gouv.fr / conseil.technique@ssi.gouv.fr

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
Premier ministre
SGDSN