Authentic 2 - Development #31862

authn OIDC : vérifier la signature de l'ID Token reçu

29 mars 2019 18:36 - Paul Marillonnet

Statut:	Fermé	**Début:**	29 mars 2019
Priorité:	Normal	**Echéance:**	
Assigné à:	Paul Marillonnet	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	Non

Description

Parce qu'en l'état -- cf https://git.entrouvert.org/authentic.git/tree/src/authentic2_auth_oidc/utils.py#n91 -- on ne la vérifie pas.

Demandes liées:

Lié à Authentic 2 - Development #26253: Fournisseur OIDC : support des signat...	**Fermé**	**08 septembre 2018**
Lié à Authentic 2 - Bug #35346: auth_oidc : l'attribut jwkset d'un fournisseu...	**Fermé**	**09 août 2019**

Révisions associées

Révision 4cc45665 - 17 octobre 2019 16:13 - Paul Marillonnet

oidc authn: verify id token signature (#31862)

Historique

#1 - 29 mars 2019 18:37 - Paul Marillonnet

- Lié à Development #26253: Fournisseur OIDC : support des signatures ECDSA ajouté

#2 - 29 mars 2019 18:47 - Paul Marillonnet

- Statut changé de Nouveau à En cours

- Assigné à mis à Paul Marillonnet

#3 - 30 mars 2019 10:02 - Benjamin Dauvergne

Pour l'instant on se contente de la sécurité apportée par le token_endpoint mais ce serait bien aussi qu'on valide la signature.

#4 - 09 août 2019 14:49 - Paul Marillonnet

Bon, j'en suis à la partie pas très drôle : j'ai ma procédure de vérification en place, qui se base sur l'implémentation de JWS par python-jwcrypto, mais qui ne fonctionne pas encore sur les jeux de données témoin présents dans les tests d'a2.

Je me creuse la tête pour savoir si (i) je m'y prends mal, en dépit des spécifications claires de vérification de la signature de l'ID Token, ou si (ii) à la lecture du code-jwcrypto, je vois que cette implémentation fait des choses hors-spécs qu'il faudrait prendre en compte ici.

#5 - 09 août 2019 15:36 - Frédéric Péters

- Lié à Bug #35346: auth_oidc : l'attribut jwkset d'un fournisseur OIDC n'est pas toujours une instance jwcrypto.JWKSet ajouté

#6 - 09 août 2019 17:42 - Benjamin Dauvergne

Je pense que tu peux tenter de remplacer toute le code parse_idtoken par la classe JWT de jwcrypto; mais il faut vérifier aussi les contraintes de validation propres à OIDC : https://openid.net/specs/openid-connect-core-1_0.html#SigEnc

#7 - 29 août 2019 17:29 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> Je pense que tu peux tenter de remplacer toute le code parse_idtoken par la classe JWT de jwcrypto; mais il faut vérifier aussi les contraintes de validation propres à OIDC : https://openid.net/specs/openid-connect-core-1_0.html#SigEnc

Oui c'est encore ce qui me paraît le plus simple. Je suis sur le coup, merci pour le tuyau.

#8 - 09 octobre 2019 14:46 - Paul Marillonnet

- Fichier 0001-oidc-authn-verify-id-token-signature-31862.patch ajouté

- Statut changé de En cours à Solution proposée

- Patch proposed changé de Non à Oui

#9 - 09 octobre 2019 14:55 - Paul Marillonnet

- Fichier 0001-oidc-authn-verify-id-token-signature-31862.patch ajouté

En virant les imports inutilisés.

#10 - 09 octobre 2019 17:09 - Benjamin Dauvergne

Il faudrait remplace ça

```
        id_token = str(id_token)
```

par

```
try:
        id_token = id_token.encode('ascii')
except UnicodeError:
    ....
```

parce que sinon ça va nous jouer des tours quand on passera en python3 mais en fait...

C'est quoi le souci qui fait que tu conserves quasiment tout le code au lieu de faire direct jwt.deserialize() ?

#11 - 09 octobre 2019 17:33 - Benjamin Dauvergne

Mon avis, d'après la doc[1], c'est que tu peux directement faire :

```
if symetric:
   algs = ['HS256', 'HS384', 'HS512']
   key = JWK(typ='oct', ...)
else:
   algs = ['RS256', ...]
   key = get_jwkset()
jwt = JWT(jwt=id_token, algs=algs, key=key)
```

[1] https://jwcrypto.readthedocs.io/en/latest/jwt.html

#12 - 09 octobre 2019 17:46 - Paul Marillonnet

- Statut changé de Solution proposée à En cours

Merci, je vais regarder tout ça.
Je crois que j'avais des embrouilles de désérialisation lorsqu'aucune clé de signature n'était mentionnée, mais je ne me souviens plus exactement du problème.
Je vais essayer à nouveau.

#13 - 09 octobre 2019 18:17 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> C'est quoi le souci qui fait que tu conserves quasiment tout le code au lieu de faire direct jwt.deserialize() ?

Alors voilà oui, (par mesure de sécurité j'imagine) l'interface de jwcrypto ne laisse pas accéder à la charge utile du jeton si la signature n'a pas été vérifiée :

```
(Pdb) jwt.claims
*** KeyError: "'claims' not set"
(Pdb) jwt.token.payload
*** InvalidJWSOperation: Payload not verified
```

Or pour vérifier la signature il faut (en plus l'identifiant de clé dans l'entête JOSE) l'identifiant de l'émetteur qui lui est fourni par la revendication 'iss' qui se trouve … dans la charge utile.

D'où le code non supprimé.
J'ai laissé aussi le test sur le nombre de parties du jeton encodé, car je voudrais par la suite y implémenter (dans un autre ticket) le support des jetons chiffrés.

#14 - 09 octobre 2019 18:21 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> Mon avis, d'après la doc[1], c'est que tu peux directement faire :

Hmm, je ne crois pas que ce soit applicable dans notre modèle de donnée. Dans le cas où l'algo de signature est symétrique, il faut connaître le client_secret associé au fournisseur OIDC ; et dans le cas où l'algo est asymétrique, il faut aussi retrouver le fournisseur OIDC pour retomber sur la bonne clé publique.

Dans tous les cas, je crois, ça impose de décoder des bouts de jeton (et d'appliquer les validations manuelles induites) pour retrouver ce fournisseur.

#15 - 09 octobre 2019 20:03 - Benjamin Dauvergne

Paul Marillonnet a écrit :

> Benjamin Dauvergne a écrit :
>
> > Mon avis, d'après la doc[1], c'est que tu peux directement faire :
>
> Hmm, je ne crois pas que ce soit applicable dans notre modèle de donnée. Dans le cas où l'algo de signature est symétrique, il faut connaître le client_secret associé au fournisseur OIDC ; et dans le cas où l'algo est asymétrique, il faut aussi retrouver le fournisseur OIDC pour retomber sur la bonne clé publique.
>
> Dans tous les cas, je crois, ça impose de décoder des bouts de jeton (et d'appliquer les validations manuelles induites) pour retrouver ce fournisseur.

Le payload est accessible via jwt.token.objects.get('payload') mais je me demande si c'est la bonne manière de faire, normalement on connaît déjà l'issuer à ce moment là (on est dans le backend, après un appel à authenticate()) il suffirait de passer le provider à authenticate.

#16 - 10 octobre 2019 10:30 - Paul Marillonnet

- Fichier 0001-oidc-authn-verify-id-token-signature-31862.patch ajouté

- Statut changé de En cours à Solution proposée

Benjamin Dauvergne a écrit :

> je me demande si c'est la bonne manière de faire, normalement on connaît déjà l'issuer à ce moment là (on est dans le backend, après un appel à authenticate()) il suffirait de passer le provider à authenticate.

Bein non, je ne crois pas. On connaît l'émetteur dans le backend parce qu'on a effectué le parsing du jeton, et on ne peut pas faire l'inverse.

Une version un peu plus courte ici, qui utilise la désérialisation sans vérification de la signature (et en conservant quelques vérifications manuelles, car cette désérialisation ne s'occupe pas du json-décodage de la charge utile).

#17 - 14 octobre 2019 09:41 - Paul Marillonnet

- Statut changé de Solution proposée à En cours

Benjamin Dauvergne a écrit :

> normalement on connaît déjà l'issuer à ce moment là (on est dans le backend, après un appel à authenticate()) il suffirait de passer le provider à authenticate.

Okay, je vois maintenant ce que tu veux dire. Ça fonctionne mais ça donne du code spaghetti. Je serais pour refactoriser en s'inspirant de l'interface des jetons jwcrypto.jwt.JWT : on implémente une fonction IDToken.deserialize qui prend en paramètre l'identifiant de l'émetteur du jeton, voire carrément l'objet OIDCProvider, et qui entre autres vérifie la signature.

#18 - 14 octobre 2019 10:17 - Benjamin Dauvergne

Paul Marillonnet a écrit :

> Okay, je vois maintenant ce que tu veux dire. Ça fonctionne mais ça donne du code spaghetti. Je serais pour refactoriser en s'inspirant de l'interface des jetons jwcrypto.jwt.JWT : on implémente une fonction IDToken.deserialize qui prend en paramètre l'identifiant de l'émetteur du jeton, voire carrément l'objet OIDCProvider, et qui entre autres vérifie la signature.

Je ne sais pas ce que tu appelles spaghetti, et oui tu peux changer toutes les APIs que tu veux, je dis juste que c'est pas la peine de retrouver une donnée qu'on a déjà et que c'est même mieux en fait (on attend un jeton de tel IdP on ne devrait pas en accepter un autre).

#19 - 15 octobre 2019 16:24 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> Je ne sais pas ce que tu appelles spaghetti, et oui tu peux changer toutes les APIs que tu veux, je dis juste que c'est pas la peine de retrouver une donnée qu'on a déjà et que c'est même mieux en fait (on attend un jeton de tel IdP on ne devrait pas en accepter un autre).

En l'état du code l'initialisateur de la classe IDToken doit recevoir d'une façon ou d'une autre un identifiant d'émetteur du jeton, et du coup l'héritage depuis le type str n'est à mon avis plus justifié. Et le cloisonnement entre parse_id_token appelé par l'initialisateur et le reste de du code ce cet initialisateur qui effectue lui aussi des vérifications sur la valeur du jeton ne l'est plus non plus.

J'ai en tête quelque chose comme :

```
token = IDToken(encoded) # <- pas de vérification à ce moment, on se contente de stocker l'encodé
try:
    token.deserialize(issuer) # <- on vérifie tout y compris la signature
except ValueError: # <- problème lors de la désérialisation ou de la vérification de la signature
    logger.error("Error on ...")
    return
claims = token.claims # <- devient accessible si la desérialisation s'est déroulée sans erreur
header = token.jose_header # <- pareil
```

#20 - 15 octobre 2019 16:35 - Benjamin Dauvergne

Tout me va.

#21 - 15 octobre 2019 17:39 - Paul Marillonnet

- Fichier 0001-oidc-authn-verify-id-token-signature-31862.patch ajouté

- Statut changé de En cours à Solution proposée

Je n'avais pas vu qu'il y a déjà ce qu'il faut dans la classe IDToken pour accéder aux claims. J'ai donc ajouté une méthode deserialize qui prend le fournisseur OIDC émetteur du jeton en argument.

#22 - 16 octobre 2019 18:47 - Benjamin Dauvergne

- Statut changé de Solution proposée à Solution validée

Paul Marillonnet a écrit :

> Je n'avais pas vu qu'il y a déjà ce qu'il faut dans la classe IDToken pour accéder aux claims. J'ai donc ajouté une méthode deserialize qui prend le fournisseur OIDC émetteur du jeton en argument.

Ok, juste je trouverai plus propre que idtoken.deserialize() lève une DeserializationError plus spécifique, pas juste une ValueError, il y a beaucoup de choses qui peuvent faire une ValueError et sur un truc beaucoup plus haut dans la chaîne d'appel je préfère qu'on fasse bien la distinction sinon ça peut cacher des soucis.

#23 - 17 octobre 2019 11:48 - Paul Marillonnet

- Fichier 0001-oidc-authn-verify-id-token-signature-31862.patch ajouté

- Statut changé de Solution validée à Solution proposée

Benjamin Dauvergne a écrit :

> Ok, juste je trouverai plus propre que idtoken.deserialize() lève une DeserializationError plus spécifique, pas juste une ValueError, il y a beaucoup de choses qui peuvent faire une ValueError et sur un truc beaucoup plus haut dans la chaîne d'appel je préfère qu'on fasse bien la distinction sinon ça peut cacher des soucis.

Ok, je comprends la nécessité de recourir à des exceptions plus spécifiques. Je propose ici une nouvelle version du patch qui reprend les exceptions de jwcrypto (et non pas DeserializationError, qui appartenant au DRF n'a à mon avis pas sa place ici).

Edit: Le patch retire la vérification de la présence et de la cohérence des claims temporelles 'exp' et 'iat', maintenant prises en charge par jwcrypto.

#24 - 17 octobre 2019 12:11 - Benjamin Dauvergne

Paul Marillonnet a écrit :

> Ok, je comprends la nécessité de recourir à des exceptions plus spécifiques. Je propose ici une nouvelle version du patch qui reprend les exceptions de jwcrypto (et non pas DeserializationError, qui appartenant au DRF n'a à mon avis pas sa place ici).

On s'est pas compris, je parlais d'une nouvelle exception pas de celle de DRF que je ne connaissais même pas :)

> Edit: Le patch retire la vérification de la présence et de la cohérence des claims temporelles 'exp' et 'iat', maintenant prises en charges par jwcrypto.

Ok.

#25 - 17 octobre 2019 12:16 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> On s'est pas compris, je parlais d'une nouvelle exception pas de celle de DRF que je ne connaissais même pas :)

Ok, j'ai dit ça après l'avoir vu utilisée dans le code de authentic.serializers.
Est-ce que le patch en l'état, réutilisant des exceptions jwcrypto plus spécifiques qu'une DeserializationError te convient ?

#26 - 17 octobre 2019 15:11 - Paul Marillonnet

- Fichier 0001-oidc-authn-verify-id-token-signature-31862.patch ajouté

Nouveau patch après échange sur le salon technique, avec une classe d'exception commune pour l'initialisation et la déserialisation du jeton.

#27 - 17 octobre 2019 16:07 - Benjamin Dauvergne

- Statut changé de Solution proposée à Solution validée

#28 - 17 octobre 2019 16:15 - Paul Marillonnet

- Statut changé de Solution validée à Résolu (à déployer)

```
commit 4cc45665b7ff5cc4c0cb1f2709609ad793cdccc3
Author: Paul Marillonnet <pmarillonnet@entrouvert.com>
Date:   Fri Aug 9 15:31:27 2019 +0200

    oidc authn: verify id token signature (#31862)
```

#29 - 07 novembre 2019 20:15 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

Fichiers

0001-oidc-authn-verify-id-token-signature-31862.patch	15 ko	09 octobre 2019	Paul Marillonnet
0001-oidc-authn-verify-id-token-signature-31862.patch	15,3 ko	09 octobre 2019	Paul Marillonnet
0001-oidc-authn-verify-id-token-signature-31862.patch	14,6 ko	10 octobre 2019	Paul Marillonnet
0001-oidc-authn-verify-id-token-signature-31862.patch	16,9 ko	15 octobre 2019	Paul Marillonnet
0001-oidc-authn-verify-id-token-signature-31862.patch	20 ko	17 octobre 2019	Paul Marillonnet
0001-oidc-authn-verify-id-token-signature-31862.patch	20 ko	17 octobre 2019	Paul Marillonnet