Authentic 2 - Development #20722

interdire l'authentification d'un compte issu d'un annuaire LDAP quand celui refuse sa connexion mais l'usager re-initialise le mot de passe

14 décembre 2017 18:06 - Serghei Mihai

Statut:	Nouveau	**Début:**	14 décembre 2017
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	

Description

Dans le cas ou un agent a oublié son mot de passe LDAP il peut le re-initialiser dans authentic et se logguer.
Or il devrait s'authentifier auprès de l'annuaire. Sinon c'est perçu comme un trou de sécurité.

Historique

#1 - 15 décembre 2017 11:02 - Paul Marillonnet

Je ne connais pas en détail la procédure de provisioning LDAP, mais si le compte n'est pas retrouvé côté annuaire, l'agent perd les groupes et rôles mappés depuis le LDAP, donc aussi les privilèges obtenus lorsque le compte était valide côté LDAP, non ?

#2 - 15 décembre 2017 15:56 - Serghei Mihai

Non.
Si un compte n'est plus dans l'annuaire rien n'est fait du côté d'Authentic: le compte reste dans la base.

#3 - 15 décembre 2017 17:04 - Paul Marillonnet

Il n'y a pas de provisioning planifié régulièrement pour vérifier que la base de comptes A2 originaires du LDAP ne diverge pas avec ce dernier ?

A la fin des fonctions populate_groups_by_mapping et populate_roles_by_mapping du LDAPBackend d'A2, on retire à un compte précédemment importé les groupes et rôles auxquels il appartient si jamais cette appartenance n'est plus valide côté LDAP.

#4 - 15 décembre 2017 17:14 - Benjamin Dauvergne

Paul Marillonnet a écrit :

> Il n'y a pas de provisioning planifié régulièrement pour vérifier que la base de comptes A2 originaires du LDAP ne diverge pas avec ce dernier ?

Oui mais ça ne fait qu'ajouter, il n'y a pas de "diff" avec le contenu dans A2 pour repérer les comptes qui ont disparu. Mais là dans un premier temps l'idée serait plus simple, lors d'une demande de ré-initialisation de mot de passe on doit traiter de façon particulière les comptes sans mot de passe et reliè à un LDAP. Actuellement un compte sans mot de passe ne pourra pas en définir via la procédure de récupération mais il pourra se connecter, au lieu de le connecter sans condition il faudrait un appel minimal vers le LDAP pour vérifier qu'il existe.

> A la fin des fonctions populate_groups_by_mapping et populate_roles_by_mapping du LDAPBackend d'A2, on retire à un compte précédemment importé les groupes et rôles auxquels il appartient si jamais cette appartenance n'est plus valide côté LDAP.

Encore faut-il que le compte existe pour arriver jusque là dans le code.

#5 - 15 décembre 2017 17:33 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> Actuellement un compte sans mot de passe ne pourra pas en définir via la procédure de récupération mais il pourra se connecter, au lieu de le connecter sans condition il faudrait un appel minimal vers le LDAP pour vérifier qu'il existe.

Donc détecter parmi les comptes sans mot de passe ceux qui proviennent du LDAP, c'est bien ça ?

> Encore faut-il que le compte existe pour arriver jusque là dans le code.

Où est le problème dans le cas où le compte n'existe pas ?

#6 - 15 décembre 2017 18:26 - Benjamin Dauvergne

Paul Marillonnet a écrit :

> Benjamin Dauvergne a écrit :
>
> > Actuellement un compte sans mot de passe ne pourra pas en définir via la procédure de récupération mais il pourra se connecter, au lieu de le connecter sans condition il faudrait un appel minimal vers le LDAP pour vérifier qu'il existe.
>
> Donc détecter parmi les comptes sans mot de passe ceux qui proviennent du LDAP, c'est bien ça ?

Voir les backends LDAPBackendPasswordLost et DummyBackend, et les vues et formulaires dans profile_views.py et profile_forms.py ; dans le cas ou le premier trouve le compte via UserExternalId mais pas dans le LDAP le deuxième ne devrait pas le retourner, je pense qu'on pourrait fusionner les deux backends en un seul ou alors faire tout ça autrement la méthode authenticate() n'étant pas l'interface la plus adaptéé pour tout ça.

Pour le code synchro lui même, en fait il faudrait remplacer le code LDAPBackend.get_users() par un vrai code de synchro qui ferait comme tout code synchro les choses en deux temps:
1. lister les comptes du coté LDAP puis créer/mettre à jour ceux dans authentic, tout en gardant une liste des external_id_tuple vues,
2. lister les comptes du coté authentic qui sont reliés à ce LDAP (User.objects.filter(userexternal__source=ldap_config['realm'])) mais donc l'id_tuple n'est pas connu (il n'est pas dans la liste accumulée au point 1) et les traiter en conséquence.

> Encore faut-il que le compte existe pour arriver jusque là dans le code.
>
> Où est le problème dans le cas où le compte n'existe pas ?

Il n'est jamais traité par le code de LDAPBackend, puisqu'aucune recherche LDAP ne le retourne.

#7 - 18 décembre 2017 10:32 - Paul Marillonnet

Je vais regarder comment je peux faire pour l'algo de synchro.

Est-ce que ça ne risque pas de taper un peu fort si on met cet algo de synchro dans LDAPBackend.get_users() ?

Et est-ce qu'il faut vraiment remplacer le code ? Je comprends la nécessité de ton algo mais je ne vois pas comment il pourrait se substituer à ce qui est déjà fait dans LDAPBackend.get_users().

(Dernière question : est-ce qu'un décorateur needs_resync placé sur toutes les méthodes qui interrogent le DN d'utilisateurs LDAP dans le backend ne pourrait pas faire l'affaire ?)

#8 - 18 décembre 2017 10:36 - Paul Marillonnet

Benjamin Dauvergne a écrit :

> 2. lister les comptes du coté authentic qui sont reliés à ce LDAP mais donc l'id_tuple n'est pas connu et les traiter en conséquence.

Est-ce qu'on laisse ce traitement configurable ?
Genre un app_settings.A2_LDAP_MISMATCH_POLICY pour laisser le choix de supprimer le compte, le détacher du LDAP, envoyer un mail de notification, ... ?

#9 - 18 décembre 2017 15:51 - Paul Marillonnet

Je me plante peut-être, mais j'ai l'impression qu'on est bien dans le cas d'usage des décorateurs, où le code ajouté ne répond pas au besoin fonctionnel, mais implémente une fonctionnalité additionnelle qui vient altérer la fonction originelle.

Peut-être qqchose comme ça du coup (du pseudo-code, qui irait dans authentic2/decorators.py) ?

```
def sync_required(function):
    def wrapped(func, *args, **kwargs):
        logger = logging.getLogger(__name__)
        # if last backend synchronization is recent:
        #     return function(request, *args, **kwargs)
        try:
            User = get_user_model()
            updated_user_list = list()
            cls = kwargs['cls']
            conn = cls.get_connection()
            for block in cls.get_config():
                user_basedn = block.get('user_basedn') or block['basedn']
                user_filter = block['sync_ldap_users_filter'] or block['user_filter']
                user_filter = user_filter.replace('%s', '*')
                attrs = cls.get_ldap_attributes_names(block)
```

```
                users = cls.paged_search(conn, user_basedn, ldap.SCOPE_SUBTREE, user_filter,
                                         attrlist=attrs)
                for user_dn, data in users:
                    # TODO A2 user base update
                    continue
                updated_user_list = [user_dn for user_dn, _ in users]

            for a2_user in User.objects.filter(userexternal__source=ldap_config['realm']):
                if not (a2_user.id_tuple & updated_user_list):
                    # TODO do something according to A2_LDAP_MISMATCH_POLICY
                    pass
        except:
            logger.error('LDAP backend synchronization operation failed.')
            return
        return function(request, *args, **kwargs)
    return wrapped
```