Authenticating a user with multiple user providers

Description

Symfony provides native support for multiple user providers. This makes it easy to integrate any kind of login handlers, including SSO and existing 3rd party bundles (e.g. FR3DLdapBundle, HWIOauthBundle, FOSUserBundle, BeSimpleSsoAuthBundle, etc.).

However, to be able to use *external* user providers with eZ, a valid eZ user needs to be injected into the repository. This is mainly for the kernel to be able to manage content-related permissions (but not limited to this).

Depending on your context, you will either want to create an eZ user `on-the-fly`, return an existing user, or even always use a generic user.

Solution

Whenever an *external* user is matched (i.e. one that does not come from eZ repository, like coming from LDAP), eZ kernel fires an `MVCEvents::INTERACTIVE_LOGIN` event. Every service listening to this event will receive an `eZ\Publish\Core\MVC\Symfony\Event\InteractiveLoginEvent` object which contains the original security token (that holds the matched user) and the request.

It's then up to the listener to retrieve an eZ user from the repository and to assign it back to the event object. This user will be injected into the repository and used for the rest of the request.

> If no eZ user is returned, the anonymous user will be used.

User exposed and security token

When an *external* user is matched, a different token will be injected into the security context, the `InteractiveLoginToken`. This token holds a `UserWrapped` instance which contains the originally matched user and the *API user* (the one from the eZ repository).

Note that the *API user* is mainly used for permission checks against the repository and thus stays *under the hood*.

Customizing the user class

It is possible to customize the user class used by extending `ezpublish.security.login_listener` service, which defaults to `eZ\Publish\Core\MVC\Symfony\Security\EventListener\SecurityListener`.

You can override `getUser()` to return whatever user class you want, as long as it implements `eZ\Publish\Core\MVC\Symfony\Security\UserInterface`.

Example

Here is a very simple example using the in-memory user provider.

In this topic:

app/config/security.yml

```
security:
    providers:
        # Chaining in_memory and ezpublish user
providers
        chain_provider:
            chain:
                providers: [in_memory, ezpublish]
        ezpublish:
            id: ezpublish.security.user_provider
        in_memory:
            memory:
                users:
                    # You will then be able to login
with username "user" and password "userpass"
                    user:  { password: userpass, roles:
[ 'ROLE_USER' ] }
 # The "in memory" provider requires an encoder for
Symfony\Component\Security\Core\User\User
 encoders:
        Symfony\Component\Security\Core\User\User:
plaintext
```

Implementing the listener

services.yml in your AcmeTestBundle

```
parameters:
    acme_test.interactive_event_listener.class:
Acme\TestBundle\EventListener\InteractiveLoginListener

services:
    acme_test.interactive_event_listener:
        class:
%acme_test.interactive_event_listener.class%
        arguments: [@ezpublish.api.service.user]
        tags:
            - { name: kernel.event_subscriber }
```

Do not mix `MVCEvents::INTERACTIVE_LOGIN` event (specific to eZ Platform) and `SecurityEvents::INTERACTIVE_LOGIN` event (fired by Symfony security component)

Interactive login listener

```
<?php
namespace Acme\TestBundle\EventListener;

use eZ\Publish\API\Repository\UserService;
use
eZ\Publish\Core\MVC\Symfony\Event\InteractiveLoginEvent;
use eZ\Publish\Core\MVC\Symfony\MVCEvents;
use
Symfony\Component\EventDispatcher\EventSubscriberInterfa
ce;

class InteractiveLoginListener implements
EventSubscriberInterface
{
    /**
     * @var \eZ\Publish\API\Repository\UserService
     */
    private $userService;

    public function __construct( UserService
$userService )
    {
        $this->userService = $userService;
    }

    public static function getSubscribedEvents()
    {
        return array(
            MVCEvents::INTERACTIVE_LOGIN =>
'onInteractiveLogin'
        );
    }

    public function onInteractiveLogin(
InteractiveLoginEvent $event )
    {
        // We just load a generic user and assign it
back to the event.
        // You may want to create users here, or even
load predefined users depending on your own rules.
        $event->setApiUser(
$this->userService->loadUserByLogin( 'lolautruche' ) );
    }
}
```