SIN6U5 : Développement web 2 (Prog. web coté serveur) Encore autour de la securité

Luigi Santocanale
LIF, Aix-Marseille Université

6 avril 2018

Plan

Sources

- Le tutoriel de Alexandre Bacco (pour Symfony 2)
- La doc de la composante `Security`
- API

Plan

Authentification et autorisation

Authentification

- A pour but de determiner l'identité du client ;
- Géré par le firewall (pare-feu) ;
- Protéger des parties du site forçant le visiteur à être un membre authentifié. Si le visiteur l'est, le firewall va le laisser passer, sinon il le redirigera sur la page d'identification.

Autorisation

- A pour but de déterminer si le visiteur (authentifié) a le droit d'accéder à la page demandée ;
- Géré par l'« access control ».

Exemple, configuration

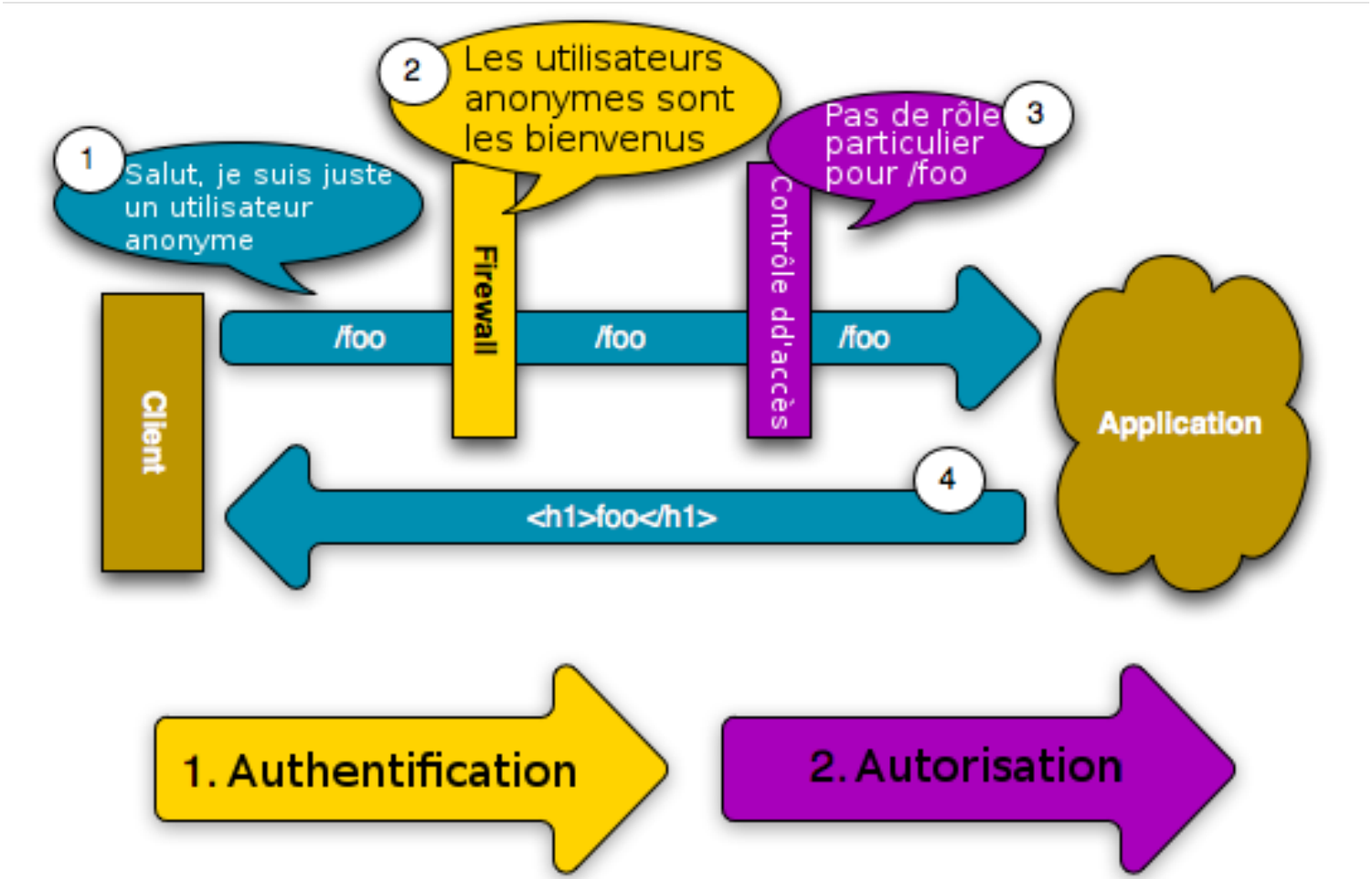
```
# app/config/security.yml
security:


    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        main:
            pattern:   ^/
            anonymous: true
            provider: in_memory
            http_basic: ~



    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
```

Exemple, configuration

```yaml
# app/config/security.yml
security:


    encoders:
        Symfony\Component\Security\Core\User\User: plaintext


    providers:
        in_memory:
            memory:
                users:
                    ryan:  { password: userpass, roles: [ 'ROLE_USER' ] }
                    admin: { password: adminpass, roles: [ 'ROLE_ADMIN' ] }


    role_hierarchy:
        ROLE_ADMIN:       ROLE_USER
        ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```

Exemple I

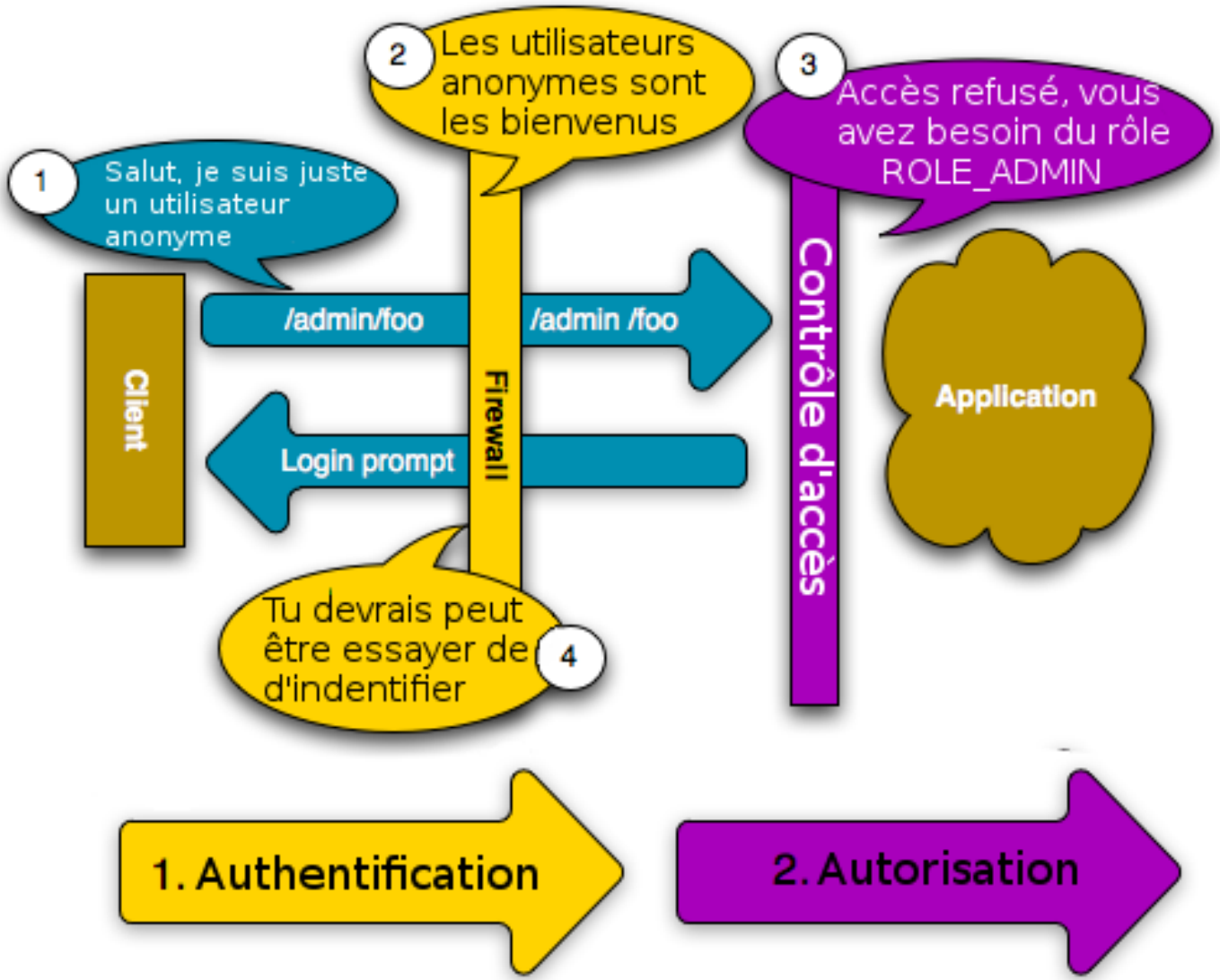
`anonyme` veut accéder à la page `/foo` qui ne requiert pas de droits.



- Le visiteur n'est pas identifié, il est `anonyme`, et tente d'accéder à la page `/foo`.
- Le firewall est configuré de telle manière qu'il n'est pas nécessaire d'être identifié pour accéder à la page `/foo`. Il laisse donc passer `anonyme`.
- Le contrôle d'accès regarde si la page `/foo` requiert des droits d'accès : il n'y en a pas. Il laisse donc passer notre visiteur, qui n'a aucun droit particulier.
- Le visiteur a donc accès à la page `/foo`.

Exemple II

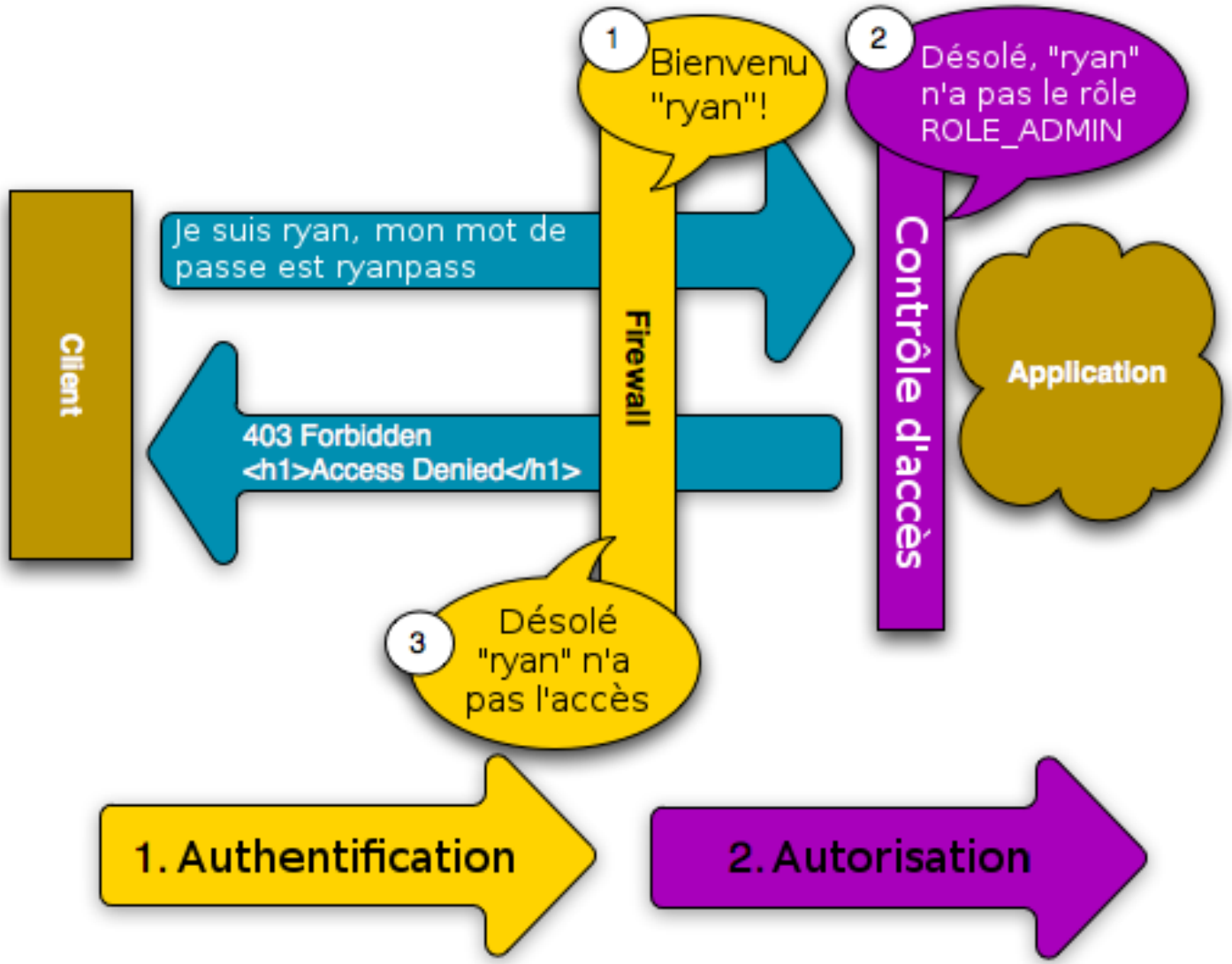
`anonyme` veut accéder à la page `/admin/foo` qui requiert certains droits.



- Le visiteur n'est pas identifié, il est toujours `anonyme`, et tente d'accéder à la page `/admin/foo`.
- Le firewall est configuré de manière qu'il ne soit pas nécessaire d'être identifié pour accéder à la page `/admin/foo`. Il laisse donc passer le visiteur.
- Le contrôle d'accès regarde si la page `/admin/foo` requiert des droits d'accès : oui, il faut le rôle `ROLE_ADMIN`. Le visiteur n'a pas ce rôle, donc le contrôle d'accès lui interdit l'accès à la page `/admin/foo`.
- Le visiteur n'a donc pas accès à la page `/admin/foo`, et se fait rédiger sur la page d'identification.

Exemple III

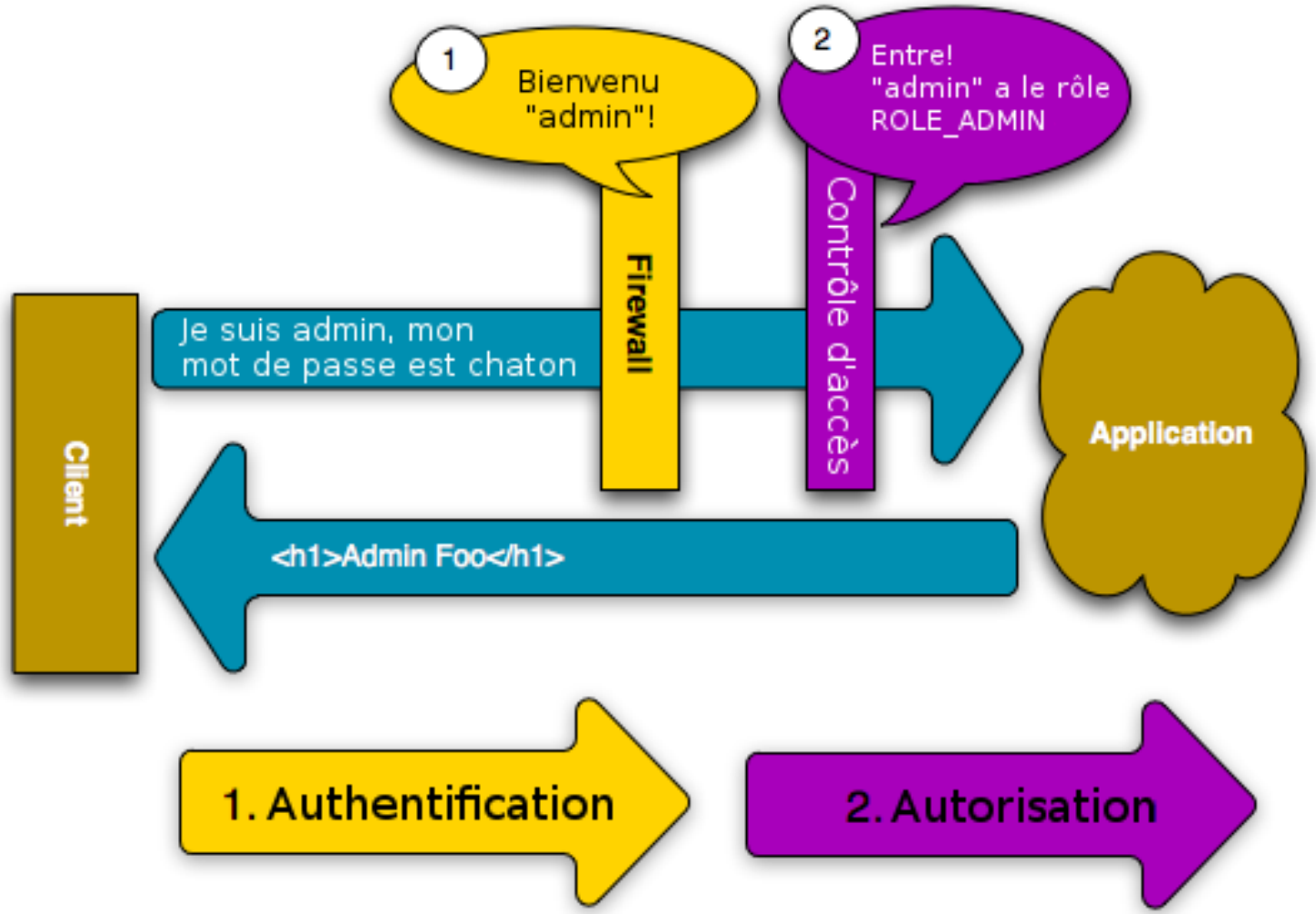
Le visiteur est identifié, veut accéder à la page `/admin/foo` qui requiert certains droits.



- `ryan` s'identifie et il tente d'accéder à la page `/admin/foo` . Le firewall confirme l'authentification de `ryan`, il laisse donc passer `ryan`.
- Le contrôle d'accès regarde si la page `/admin/foo` requiert des droits d'accès : oui, il faut le rôle `ROLE_ADMIN`. `ryan` n'a pas ce rôle et le contrôle d'accès lui interdit l'accès à la page `/admin/foo`.
- Le contrôle d'accès lui affiche donc une page d'erreur lui disant qu'il n'a pas les droits suffisants.

Exemple IV

Je suis identifié, et je veux accéder à la page `/admin/foo` qui requiert des droits que j'ai.



- L'utilisateur `admin` s'identifie et tente d'accéder à la page `/admin/foo` . Le firewall confirme l'authentification d'`admin` et il le laisse donc passer.
- Le contrôle d'accès regarde si la page `/admin/foo` requiert des droits d'accès : oui, il faut le rôle `ROLE_ADMIN`. `admin` a ce rôle et le contrôle d'accès le laisse passer.
- L'utilisateur `admin` a accès à la page `/admin/foo`.

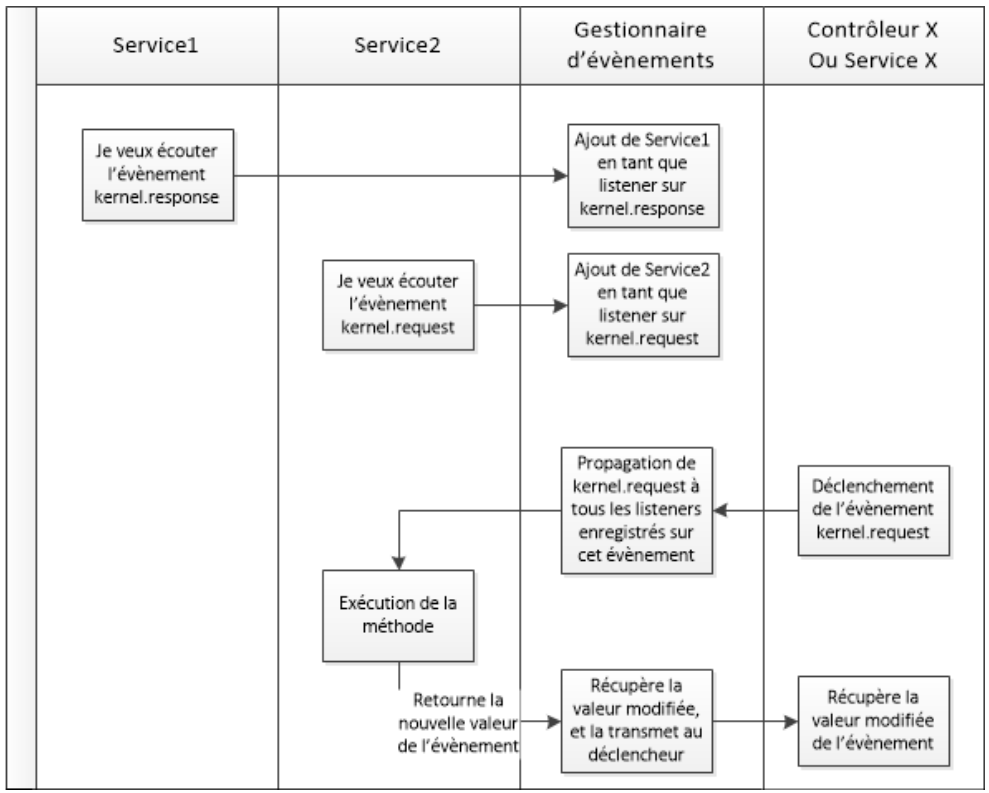
Processus général

- Un utilisateur veut accéder à une ressource protégée ;
- Le firewall redirige l'utilisateur au formulaire de connexion ;
- L'utilisateur soumet ses informations d'identification (par exemple login et mot de passe) ;
- Le firewall authentifie l'utilisateur ;
- L'utilisateur authentifié renvoie la requête initiale ;
- Le contrôle d'accès vérifie les droits de l'utilisateur, et autorise ou non l'accès à la ressource protégée.

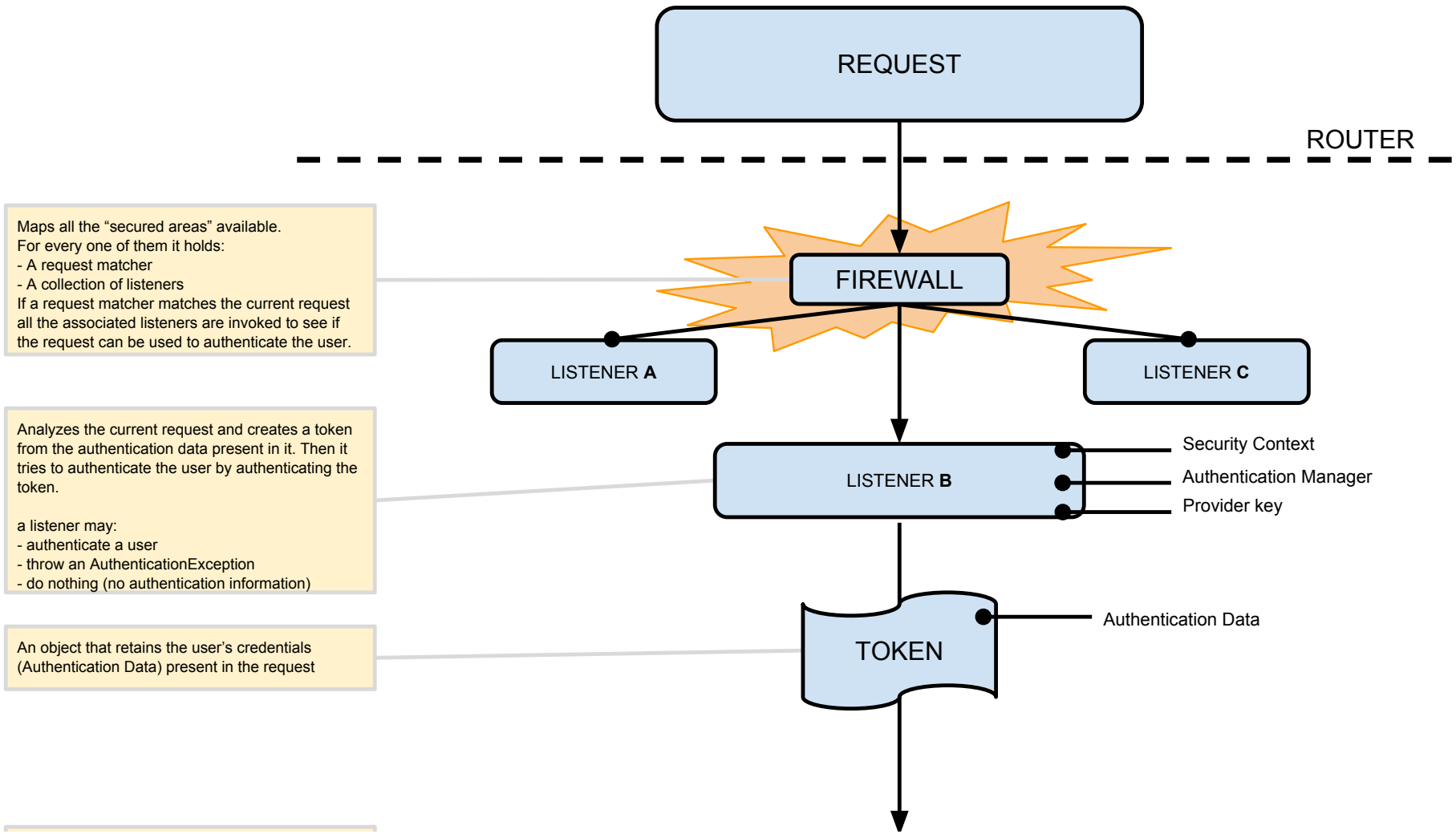
Plan

Les événements en Symfony

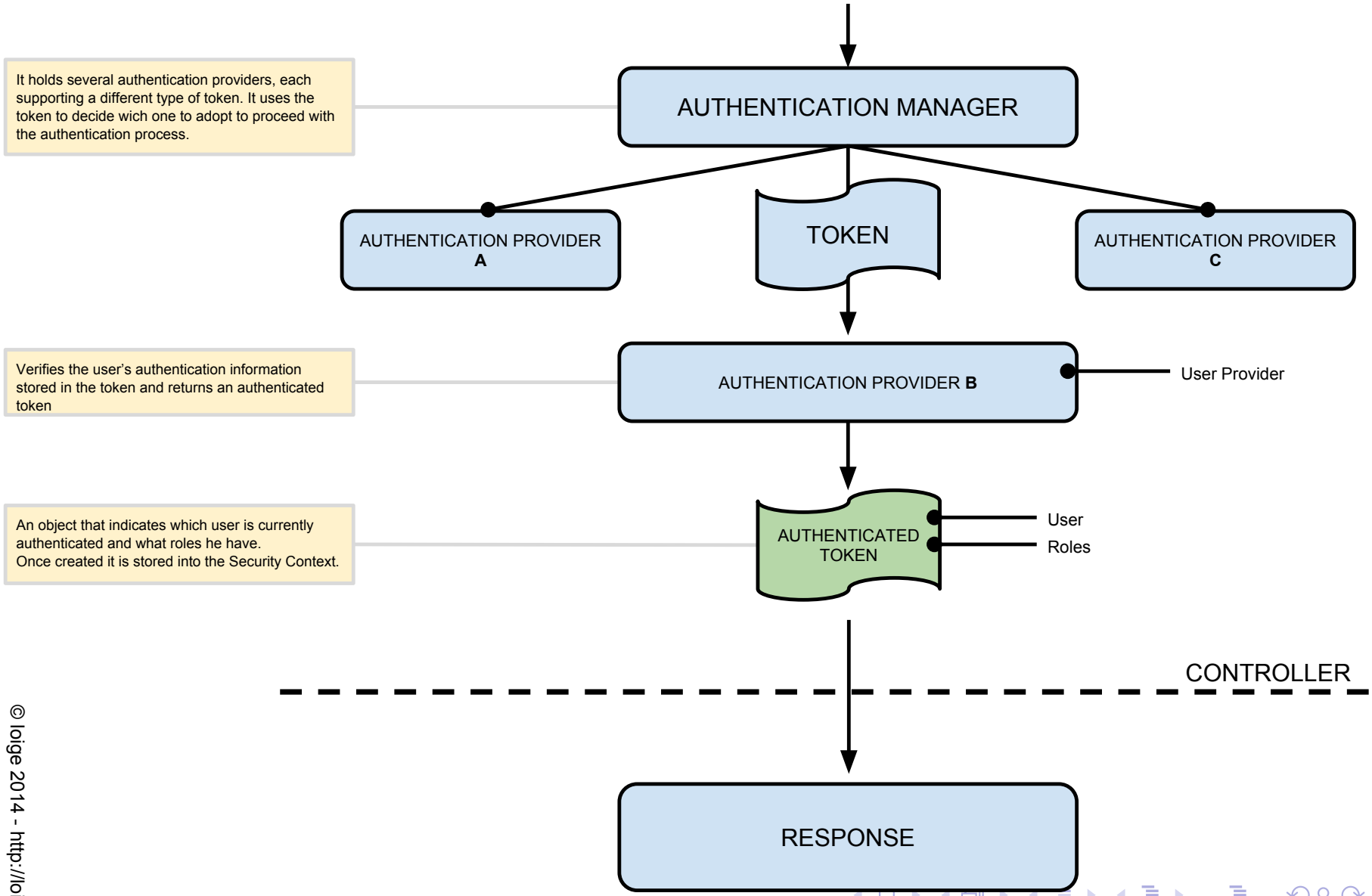
Le tutoriel de Alexandre Bacco sur les événements.



Le pare-feu, schéma Symfony I



Le firewall, schéma Symfony II



Le pare-feu

Mappe de toute area protegée disponible.
Dans notre exemple nous avons deux areas :

- `^/(_(profiler|wdt)|css|images|js)/`,
- `^/`

Pour chaque area protegée il possède :

- Un objet « request matcher »
- Une collection d'écouteurs (« listeners »)

Si une requête est filtrée, alors tous ses écouteurs sont invoquées.

```php
use Symfony\Component\Security\Http\FirewallMap;
use Symfony\Component\HttpFoundation\RequestMatcher;
use Symfony\Component\Security\Http\Firewall\ExceptionListener;

$firewallMap = new FirewallMap();

$requestMatcher = new RequestMatcher('^/secured-area/');

// instances of
  ↪   Symfony\Component\Security\Http\Firewall\ListenerInterface
$listeners = array(...);

$exceptionListener = new ExceptionListener(...);

$firewallMap->add($requestMatcher, $listeners, $exceptionListener);
```

Le pare-feu devient un écouteur de l'évènement `kernel.request`

```php
use Symfony\Component\Security\Http\Firewall;
use Symfony\Component\HttpKernel\KernelEvents;

// the EventDispatcher used by the HttpKernel
$dispatcher = ...;

$firewall = new Firewall($firewallMap, $dispatcher);

$dispatcher->addListener(
    KernelEvents::REQUEST,
    array($firewall, 'onKernelRequest')
);
```

Les écouteurs du pare-feu

- Analyse la requête courante.
- Crée un jeton à partir des données d'authentification présentes dans la requête (session, cookie, POST . . .).

  Exemple : l'écouteur « Basic HTTP authentification » vérifie si la requête possède un header nommé `PHP_AUTH_USER`.
- Il utilise ensuite le jeton pour authentifier l'utilisateur.
- Il peut ainsi :
  - authentifier l'utilisateur ;
  - lever une exception `AuthenticationException` ;
  - rien faire (ne donner aucune information d'authentification).

Les écouteurs des exceptions

Si un des écouteurs a levé l'exception

`AuthenticationException`,

l'écouteur des exception fourni (lors de l'ajout d'une area securisée) prendra le relais.

Il détermine que va se passer à la suite. Il peut :

- démarrer la procédure d' authentification,
- demander à l'utilisateur de rentrer ses cordonnées une autre fois (quand on a authentifié par un "remember-me", ou bien
- transformer l'exception en une `AccessDeniedHttpException`, donnant à la fin une réponse "HTTP/1.1 403 : Access Denied".

Point d'éntrée

Si l'utilisateur n'est pas authentifié, le point d'entrée du pare-feu est appelé.

Le point d'entrée prend en paramètre un objet de la classe `Request` et l'exception levée et retourne un objet de la classe `Response`.
Par exemple :

- la page contenant un un formulaire authentification ou,
- une réponse avec un header `WWW-Authenticate`, dans le cas du Basic HTTP.

L'« Authentication manager »

- Il possède plusieurs fournisseurs (« providers »), chacun supportant un type diffèrent de jeton.
- Il utilise le jeton pour décider quel fournisseur utiliser pour le processus d'authentification.

```php
use Symfony\Component\Security\Core\Authentication\AuthenticationProviderManager;
use Symfony\Component\Security\Core\Exception\AuthenticationException;

// instances of
  ↪    Symfony\Component\Security\Core\Authentication\Provider\AuthenticationProviderInterface
$providers = array(...);

$authenticationManager = new AuthenticationProviderManager($providers);

try {
    $authenticatedToken = $authenticationManager
        ->authenticate($unauthenticatedToken);
} catch (AuthenticationException $exception) {
    // authentication failed
}
```

Le fournisseur d'authentification et le jeton authentifié

- Vérifie les informations d'authentification de l'utilisateur stockées dans le jeton et
- retourne un jeton authentifié, ou bien
- lève l'exception `AuthenticationException` (ou des extensions).
- Le jeton authentifié : objet témoignant que l'utilisateur est authentifié et quel sont ses rôles.
- Une fois créé, il est stocké dans le contexte de sécurité.

Plan

Exemple utilisation DAO provider

```php
use Symfony\Component\Security\Core\Authentication\Provider\DaoAuthenticationProvider;
use Symfony\Component\Security\Core\User\UserChecker;
use Symfony\Component\Security\Core\User\InMemoryUserProvider;
use Symfony\Component\Security\Core\Encoder\EncoderFactory;

$userProvider = new InMemoryUserProvider(
    array(
        'admin' => array(
            // password is "foo"
            'password' =>
 ↪    '5FZ2Z8QIkA7UTZ4BYkoC+GsReLf569mSKDsfods6LYQ8t+a8EW9oaircfMpmaLbPBh4FOBiiFyLfuZmTSUwzZg==',
            'roles'    => array('ROLE_ADMIN'),
        ),
    )
);

// for some extra checks: is account enabled, locked, expired, etc.
$userChecker = new UserChecker();

// an array of password encoders (see below)
$encoderFactory = new EncoderFactory(...);

$daoProvider = new DaoAuthenticationProvider(
    $userProvider,
    $userChecker,
    'secured_area',
    $encoderFactory
);

$daoProvider->authenticate($unauthenticatedToken);
```

L'interface AuthenticationProviderInterface

```php
<?php


namespace Symfony\Component\Security\Core\Authentication\Provider;

use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Authentication\AuthenticationManagerInterface;



interface AuthenticationProviderInterface extends AuthenticationManagerInterface
{
    /**
     * Use this constant for not provided username.
     *
     * @var string
     */
    const USERNAME_NONE_PROVIDED = 'NONE_PROVIDED';

    /**
     * Checks whether this provider supports the given token.
     *
     * @return bool true if the implementation supports the Token, false otherwise
     */
    public function supports(TokenInterface $token);
}
```

L'interface AuthenticationManagerInterface

```php
<?php



namespace Symfony\Component\Security\Core\Authentication;

use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Exception\AuthenticationException;



interface AuthenticationManagerInterface
{
    /**
     * Attempts to authenticate a TokenInterface object.
     *
     * @param TokenInterface $token The TokenInterface instance to authenticate
     *
     * @return TokenInterface An authenticated TokenInterface instance, never null
     *
     * @throws AuthenticationException if the authentication fails
     */
    public function authenticate(TokenInterface $token);
}
```

Le UserAuthenticationProvider

```php
<?php

/*
 * This file is part of the Symfony package.
 *
 * (c) Fabien Potencier <fabien@symfony.com>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Symfony\Component\Security\Core\Authentication\Provider;

use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\User\UserCheckerInterface;
use Symfony\Component\Security\Core\Exception\UsernameNotFoundException;
use Symfony\Component\Security\Core\Exception\AuthenticationException;
use Symfony\Component\Security\Core\Exception\BadCredentialsException;
use Symfony\Component\Security\Core\Exception\AuthenticationServiceException;
use Symfony\Component\Security\Core\Authentication\Token\UsernamePasswordToken;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
```

Le UserAuthenticationProvider

```php
abstract class UserAuthenticationProvider implements AuthenticationProviderInterface
{
    private $hideUserNotFoundExceptions;
    private $userChecker;
    private $providerKey;

    /**
     * @param UserCheckerInterface $userChecker                An UserCheckerInterface
    ↪    interface
     * @param string               $providerKey                A provider key
     * @param bool                 $hideUserNotFoundExceptions Whether to hide user not found
    ↪    exception or not
     *
     * @throws \InvalidArgumentException
     */
    public function __construct(UserCheckerInterface $userChecker, $providerKey,
    ↪    $hideUserNotFoundExceptions = true)
    {
        if (empty($providerKey)) {
            throw new \InvalidArgumentException('$providerKey must not be empty.');
        }

        $this->userChecker = $userChecker;
        $this->providerKey = $providerKey;
        $this->hideUserNotFoundExceptions = $hideUserNotFoundExceptions;
    }
```

Le UserAuthenticationProvider

```php
    public function authenticate(TokenInterface $token)
    {
        if (!$this->supports($token)) {
            throw new AuthenticationException('The token is not supported by this authentication provider.');
        }

        $username = $token->getUsername();
        if ('' === $username || null === $username) {
            $username = AuthenticationProviderInterface::USERNAME_NONE_PROVIDED;
        }

        try {
            $user = $this->retrieveUser($username, $token);
        } catch (UsernameNotFoundException $e) {
            if ($this->hideUserNotFoundExceptions) {
                throw new BadCredentialsException('Bad credentials.', 0, $e);
            }
            $e->setUsername($username);

            throw $e;
        }

        if (!$user instanceof UserInterface) {
            throw new AuthenticationServiceException('retrieveUser() must return a UserInterface.');
        }

        try {
            $this->userChecker->checkPreAuth($user);
            $this->checkAuthentication($user, $token);
            $this->userChecker->checkPostAuth($user);
        } catch (BadCredentialsException $e) {
            if ($this->hideUserNotFoundExceptions) {
                throw new BadCredentialsException('Bad credentials.', 0, $e);
            }

            throw $e;
        }

        $authenticatedToken = new UsernamePasswordToken($user, $token->getCredentials(), $this->providerKey,
↪       $this->getRoles($user, $token));
        $authenticatedToken->setAttributes($token->getAttributes());

        return $authenticatedToken;
    }
```

Le UserAuthenticationProvider

```
103     public function supports(TokenInterface $token)
104     {
105         return $token instanceof UsernamePasswordToken && $this->providerKey ===
    ↪    $token->getProviderKey();
106     }
107
108     /**
109      * Retrieves roles from user and appends SwitchUserRole if original token contained one.
110      *
111      * @return array The user roles
112      */
113     private function getRoles(UserInterface $user, TokenInterface $token)
114     {
115         $roles = $user->getRoles();
116
117         foreach ($token->getRoles() as $role) {
118             if ($role instanceof SwitchUserRole) {
119                 $roles[] = $role;
120
121                 break;
122             }
123         }
124
125         return $roles;
126     }
```

Le UserAuthenticationProvider

```
128     /**
129      * Retrieves the user from an implementation-specific location.
130      *
131      * @param string                $username The username to retrieve
132      * @param UsernamePasswordToken $token    The Token
133      *
134      * @return UserInterface The user
135      *
136      * @throws AuthenticationException if the credentials could not be validated
137      */
138     abstract protected function retrieveUser($username, UsernamePasswordToken $token);
139
140     /**
141      * Does additional checks on the user and token (like validating the
142      * credentials).
143      *
144      * @throws AuthenticationException if the credentials could not be validated
145      */
146     abstract protected function checkAuthentication(UserInterface $user, UsernamePasswordToken
  ↪    $token);
147 }
```

Le DAO provider

```php
<?php

/*
 * This file is part of the Symfony package.
 *
 * (c) Fabien Potencier <fabien@symfony.com>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Symfony\Component\Security\Core\Authentication\Provider;

use Symfony\Component\Security\Core\Encoder\EncoderFactoryInterface;
use Symfony\Component\Security\Core\User\UserProviderInterface;
use Symfony\Component\Security\Core\User\UserCheckerInterface;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\Exception\UsernameNotFoundException;
use Symfony\Component\Security\Core\Exception\AuthenticationServiceException;
use Symfony\Component\Security\Core\Exception\BadCredentialsException;
use Symfony\Component\Security\Core\Authentication\Token\UsernamePasswordToken;
```

Le DAO provider

```php
/**
 * DaoAuthenticationProvider uses a UserProviderInterface to retrieve the user
 * for a UsernamePasswordToken.
 *
 * @author Fabien Potencier <fabien@symfony.com>
 */
class DaoAuthenticationProvider extends UserAuthenticationProvider
{
    private $encoderFactory;
    private $userProvider;

    /**
     * @param UserProviderInterface   $userProvider              An UserProviderInterface instance
     * @param UserCheckerInterface    $userChecker               An UserCheckerInterface instance
     * @param string                  $providerKey               The provider key
     * @param EncoderFactoryInterface $encoderFactory            An EncoderFactoryInterface instance
     * @param bool                    $hideUserNotFoundExceptions Whether to hide user not found exception or not
     */
    public function __construct(UserProviderInterface $userProvider, UserCheckerInterface $userChecker,
    ↪    $providerKey, EncoderFactoryInterface $encoderFactory, $hideUserNotFoundExceptions = true)
    {
        parent::__construct($userChecker, $providerKey, $hideUserNotFoundExceptions);

        $this->encoderFactory = $encoderFactory;
        $this->userProvider = $userProvider;
    }
```

Le DAO provider

```
52      protected function checkAuthentication(UserInterface $user, UsernamePasswordToken $token)
53      {
54          $currentUser = $token->getUser();
55          if ($currentUser instanceof UserInterface) {
56              if ($currentUser->getPassword() !== $user->getPassword()) {
57                  throw new BadCredentialsException('The credentials were changed from another session.');
58              }
59          } else {
60              if ('' === ($presentedPassword = $token->getCredentials())) {
61                  throw new BadCredentialsException('The presented password cannot be empty.');
62              }
63
64              if (!$this->encoderFactory->getEncoder($user)->isPasswordValid($user->getPassword(),
    ↪     $presentedPassword, $user->getSalt())) {
65                  throw new BadCredentialsException('The presented password is invalid.');
66              }
67          }
68      }
```

Le DAO provider

```
73      protected function retrieveUser($username, UsernamePasswordToken $token)
74      {
75          $user = $token->getUser();
76          if ($user instanceof UserInterface) {
77              return $user;
78          }
79
80          try {
81              $user = $this->userProvider->loadUserByUsername($username);
82
83              if (!$user instanceof UserInterface) {
84                  throw new AuthenticationServiceException('The user provider must return a UserInterface object.');
85              }
86
87              return $user;
88          } catch (UsernameNotFoundException $e) {
89              $e->setUsername($username);
90              throw $e;
91          } catch (\Exception $e) {
92              $e = new AuthenticationServiceException($e->getMessage(), 0, $e);
93              $e->setToken($token);
94              throw $e;
95          }
96      }
97  }
```