Mastering Symfony2

SF2C2

SensioLabs

Chapter 4

Security

The Security Component and Bundle

Provides **Authentication** and **Authorization** mechanisms

Authentication ensures the user’s identity.

Authorization grants or denies a user to perform an action.

Some key concepts…

Tool	Meaning
encoder	Hashes and compares passwords
provider	Finds and / or creates users
firewall	Sets the authentication mechanism for each application part
access_control	Secures parts of the application with roles

Encoding and checking passwords

An **encoder** is used for hashing and comparing passwords

The PasswordEncoderInterface interface

```php
namespace Symfony\Component\Security\Core\Encoder;

interface PasswordEncoderInterface
{
    function encodePassword($raw, $salt);

    function isPasswordValid($encoded, $raw, $salt);
}
```

Configuring encoders

```
# app/config/security.yml

security:
    encoders:
        AppBundle\Entity\User: plaintext # dev only
        AppBundle\Entity\User: bcrypt
        AppBundle\Entity\Player:
            algorithm:         bcrypt
            cost:              13
        AppBundle\Entity\Admin:
            id: app.custom_service
```

When PHP < 5.5 and using bcrypt:

`composer require “ircmaxell/password_compat”`

Choosing the right salt

It's recommended to generate a different and random salt value for each application user.

Bad practice

```
// generate a basic time-based salt
$salt = md5(time());
```

Best practice

```
// generate a complex random salt
$salt = base_convert(sha512(uniqid(mt_rand(), true)), 16,
36);
```

Encoding a password

```php
$user = new AppBundle\Entity\User();

$salt = $this->get('security.secure_random')->nextBytes(15);
$user->setSalt(md5($salt));

// since 2.6
$encoder = $this->get('security.password_encoder');
$pwd = $encoder->encodePassword($user, 'my-pwd');

$user->setPassword($pwd);
```

Checking the validity of the password

```php
$encoder->isPasswordValid('e5e[...]ca5', 'secret', '^H4x0r$');
```

Fetching users on a user provider

A **provider** knows how to retrieve a user.

Built-in user providers in Symfony SE

Type	Meaning
memory	Fetches users from a configuration file (security.yml)
chain	Fetches users by chaining multiple providers
entity	Fetches users from a Doctrine entity model
propel	Fetches users from a Propel active record model

```php
interface UserProviderInterface
{
    /**
     * Loads the user for the given username.
     *
     * @return UserInterface
     */
    function loadUserByUsername($username);

    /**
     * Refreshes the user properties like credentials.
     *
     * @return UserInterface
     */
    function refreshUser(UserInterface $user);

    /**
     * Whether this provider supports the given user class
     *
     * @return Boolean
     */
    function supportsClass($class);
}
```

Storing users in memory

```
security:
    providers:
        administrators:
            memory:
                users:
                    jsmith:
                        password: hashed_secret
                        roles: [ 'ROLE_USER' ]
                    hhamon:
                        password: hashed_azerty
                        roles: [ 'ROLE_TRAINER' ]
                    fabpot:
                        password: hashed_qwerty
                        roles: ['ROLE_TRAINER', 'ROLE_ADMIN' ]
```

Representing a user in the security layer

A **user** object stores **credentials** and associated **roles**.

The UserInterface interface

```
interface UserInterface
{
    function getRoles();

    function getPassword();

    function getSalt();

    function getUsername();

    function eraseCredentials();
}
```

The AdvancedUserInterface interface

```
interface AdvancedUserInterface extends UserInterface
{
    function isEnabled();

    function isCredentialsNonExpired();

    function isAccountNonLocked();

    function isAccountNonExpired();
}
```

Managing users’ roles

Roles are strings but they can be any objects of type **RoleInterface**.

The RoleInterface interface

```
interface RoleInterface
{
    /**
     * Returns the role name.
     *
     * @return string The role name
     */
    function getRole();
}
```

The roles hierarchy

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

        ROLE_ADMIN:       ROLE_USER
        ROLE_TRAINER:     ROLE_USER
        ROLE_SUPERADMIN:
            - ROLE_USER
            - ROLE_ADMIN
```

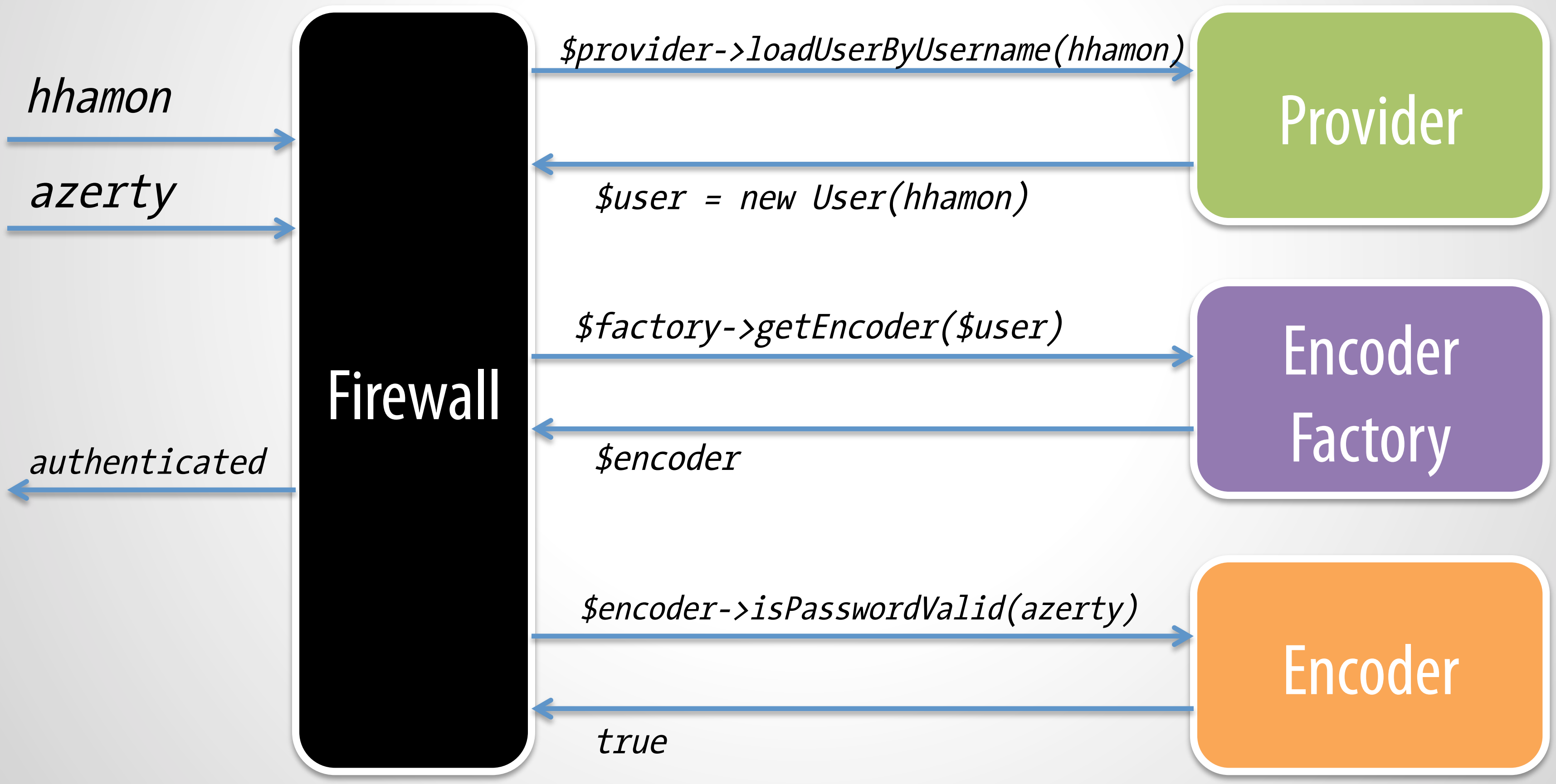
Authenticating against a firewall

The **firewall** determines whether or not the user needs to be authenticated.

Supported authentication firewalls

Type	Meaning
http_basic	Use basic HTTP authentication
http_digest	Use basic HTTP authentication with a hashed password
x509	Use a x.509 certificate
form_login	Use a simple web form to ask for the login and password credentials

Authentication worflow

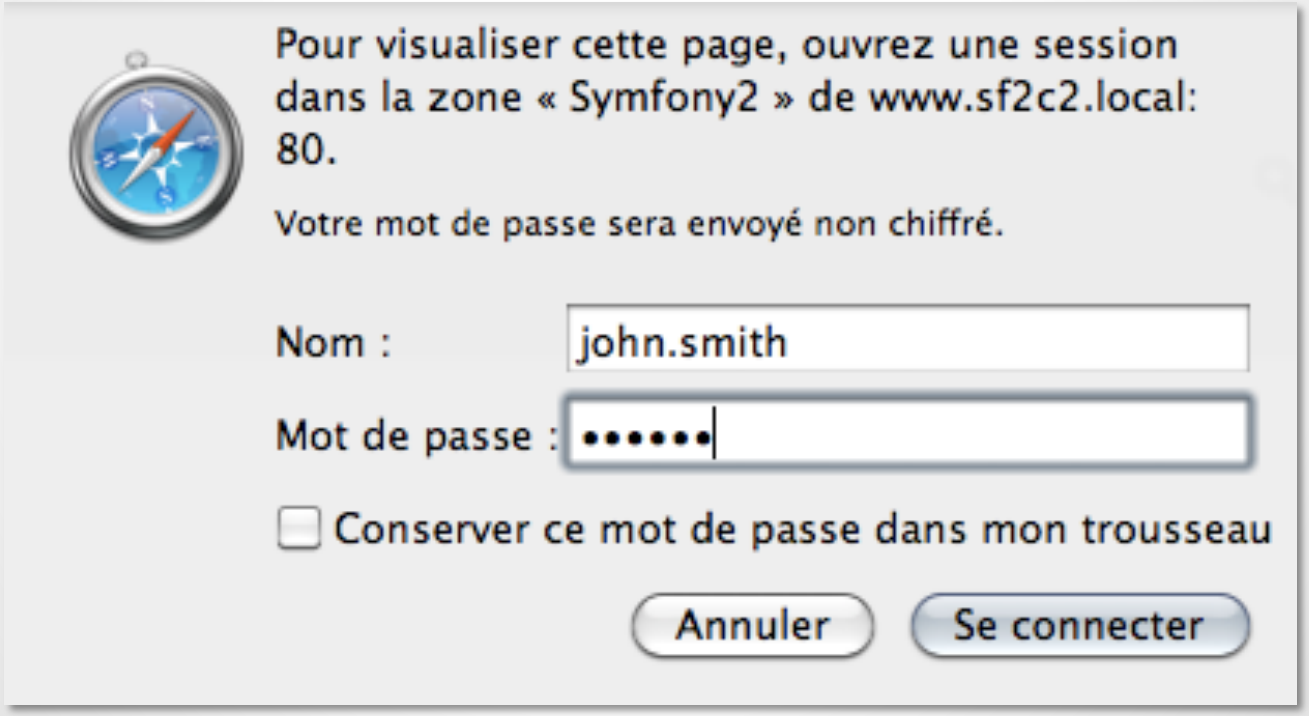


Configuring an HTTP Authentication

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

    # ...
    firewalls:
        admin:
            provider:   administrators
            pattern:    ^/admin
            http_basic:
                realm:   "Symfony2"
```

Authenticating with HTTP Basic


Pour visualiser cette page, ouvrez une session dans la zone « Symfony2 » de www.sf2c2.local: 80.

Votre mot de passe sera envoyé non chiffré.

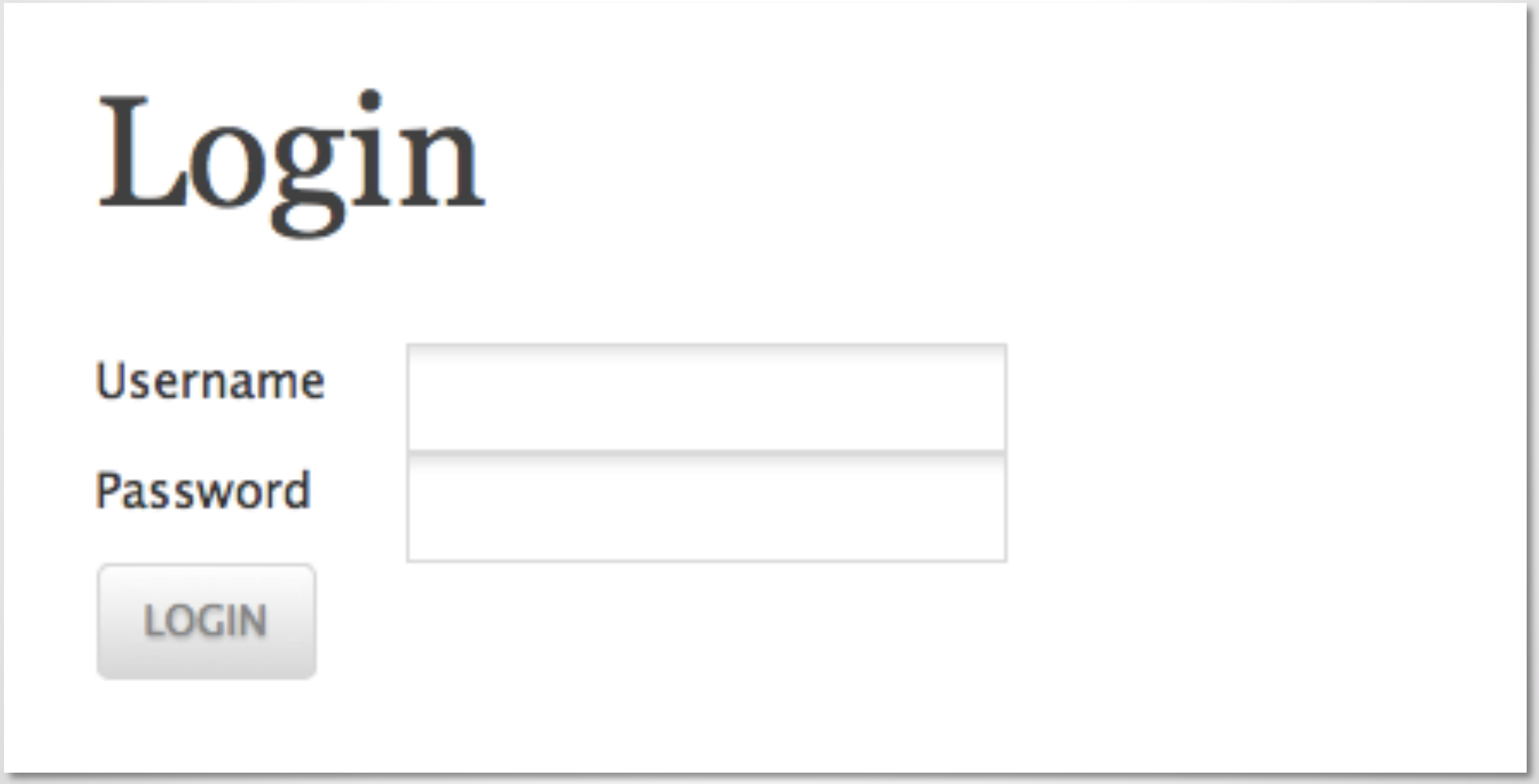
Nom : john.smith

Mot de passe : ••••••

Authenticating with a login form

```
# app/config/security.yml
security:
    # ...
    firewalls:
        admin:
            provider:                           administrators
            pattern:                            ^/admin
            form_login:
                check_path:                     login_check
                login_path:                     signin
                default_target_path:            admin
                always_use_default_target_path: true
```

The login form



Adding routes for authentication

```
# app/config/routing.yml

login_check:
    path:         /admin/auth
    methods:      POST

signout:
    path:         /admin/logout
    methods:      GET

signin:
    path:         /login
    defaults:     { _controller: AppBundle:User:signin }
```

```php
use Symfony\Component\HttpFoundation\Request;

class SecurityController extends Controller
{
    public function loginAction(Request $request)
    {
        // since 2.6
        $helper = $this->get('security.authentication_utils');

        $name = $helper->getLastUsername();
        $error = $helper->getLastAuthenticationError();

        return $this->render(
            'user/signin.html.twig',
            array('last_username' => $name, 'error' => $error)
        );
    }
}
```

```
{% extends 'SensioUserBundle::layout.html.twig' %}

{% block content %}
    <h1>Login</h1>

    {% if error %}
        <div class="error">{{ error.message }}</div>
    {% endif %}

    <form action="{{ path("login_check") }}" method="post">
        <div>
            <label for="username">Username</label>
            <input type="text" id="username"
                   name="_username" value="{{ last_username }}" />
        </div>
        <div>
            <label for="password">Password</label>
            <input type="password" id="password" name="_password" />
        </div>

        <button type="submit">login</button>
    </form>
{% endblock %}
```

Allowing anonymous authentication

```yaml
# app/config/security.yml
security:
    # ...
    firewalls:
        frontend:
            # ...
            pattern:   ^/
            anonymous: true
```

Allowing logout capability

```
# app/config/security.yml
security:
    # ...
    firewalls:
        admin:
            # ...
            logout:
                path:   signout
                target: signin
```

Accessing the user from a controller

```
$user = $this->getUser();
```

Accessing the user from a template

```
{{ app.user.username }}
```

Allowing admin users to switch context

```
# app/config/security.yml
security:
    # ...
    firewalls:
        frontend:
            # ...
            pattern:   ^/
            switch_user: true
```

Allowing admin users to switch context

```yaml
security:
    providers:
        administrators:
            memory:
                users:
                    superadmin:
                        password: superman
                        roles:
                            - 'ROLE_ADMIN'
                            - 'ROLE_ALLOWED_TO_SWITCH'
                    ...
```

Switching to another security user

http://my.domain.com/app_dev.php/admin?_switch_user=hhamon

Forcing the security token

```
$token = new UsernamePasswordToken(
    'hhamon',
    'p4SSw0rD',
    'administrators',
    array('ROLE_ADMIN')
);

$this
    ->get('security.token_storage')
    ->setToken($token);
```

Supported authentication tokens

Class	Meaning
AnonymousToken	Token for anonymous users.
RememberMeToken	Used when authenticating with a remember me cookie.
PreAuthenticatedToken	Used when requests are already pre-authenticated.
UsernamePasswordToken	Used when authenticating with a username and password.
PersistentToken	Used when authenticating with a cookie.

Controlling access to application resources

Access control rules

secure parts of the application according to user’s roles.

Securing the application with roles

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

    access_control:
        -
            path: ^/admin
            roles: [ROLE_ADMIN, ROLE_MANAGER]
        -
            path: ^/account
            roles: [ROLE_USER]
```

Securing the application with roles

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

  access_control:
    -
      path: ^/admin
      host: mydomain.foo
      ip: 192.0.0.0/8
      requires_channel: https
      condition: "request.headers.get('User-Agent') matches /Firefox/i"
      roles: [ROLE_ADMIN, ROLE_MANAGER]
```

Granting or denying access to the user

```
public function editAction($id)
{
    // since 2.6
    $security = $this->get('security.authorization_checker');

    // check for edit permission
    if (!$security->isGranted(['ROLE_ADMIN'])) {

        throw new AccessDeniedException([...]);
    }

    // granted to perform an action...
}
```

Granting or denying access to the user

```php
public function editAction($id)
{
    // since 2.6
    $this->denyAccessUnlessGranted('ROLE_ADMIN');

    // granted to perform an action...
}
```

Granting or denying access to the user

```
use Sensio[…]ExtraBundle\Configuration\Security;

/**
 * @Route("/admin/user/{id}")
 * @Security("has_role('ROLE_ADMIN')")
 */
public function editAction($id)
{
    // granted to perform an action...
}
```

Checking roles from a Twig template

```
{% if is_granted("ROLE_ADMIN") %}

    <a href="...">Delete</a>

{% endif %}
```

SensioLabs Training Department

Address

92-98 Boulevard Victor Hugo

92 115 Clichy Cedex

France

Phone

+33 140 998 205

Email

training@sensiolabs.com

SensioLabs