Security

The Security Component and Bundle

The Security Component and Bundle

- Provides **Authentication** & **Authorization** mechanisms.
- **Authentication** ensures the user's identity. (Who are you?)
- **Authorization** grants or denies a user to perform an action. (Can you do that?)

Some key concepts…

Method	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

Password encoder

A **password encoder** is used to hash and compare a raw password.

The password encoder interfaces

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

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

interface UserPasswordEncoderInterface
{
    function encodePassword(UserInterface $user, $plainPassword);
    function isPasswordValid(UserInterface $user, $rawPassword);
}
```

Configuring password encoders

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

Choosing the right salt

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

This is not needed when using **bcrypt** (the salt is generated automatically).

```
$user->setSalt(sha1(random_bytes()));
```

Encoding a password

The SecurityBundle comes with a built-in user password encoder to encode a raw password with the right configured password encoder.

```
$encoder = $this->get('security.password_encoder');
$password = $encoder->encodePassword(
    $user, 'my_password'
);
$user->setPassword($password);
```

Checking the validity of the password

The SecurityBundle comes with a built-in user password encoder to check if a raw password is valid.

```
$encoder = $this->get('security.password_encoder');
$result = $encoder->isPasswordValid(
    $user, 'my_password'
);
```

Fetching users with a user provider

User Providers

- When a user submits a username and password, the authentication layer asks the configured **user provider** to **return a user object** for a given username.
- Symfony has two built-in user providers: "in_memory" and "entity".

Built-in user providers in Symfony SE

Method	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

The UserProvider interface

```
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

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

The User implementation

A user implementation stores **credentials** and its associated **roles** or **permissions**.

The UserInterface interface

```
interface UserInterface
{
    public function getRoles();
    public function getPassword();
    public function getSalt();
    public function getUsername();
    public function eraseCredentials();
}
```

The AdvancedUserInterface interface

```
interface AdvancedUserInterface
          extends UserInterface
{
    function isEnabled();
    function isCredentialsNonExpired();
    function isAccountNonLocked();
    function isAccountNonExpired();
}
```

Managing user's roles

Security roles

Roles are strings by default but they can also be defined as objects implementing the **RoleInterface** interface.

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

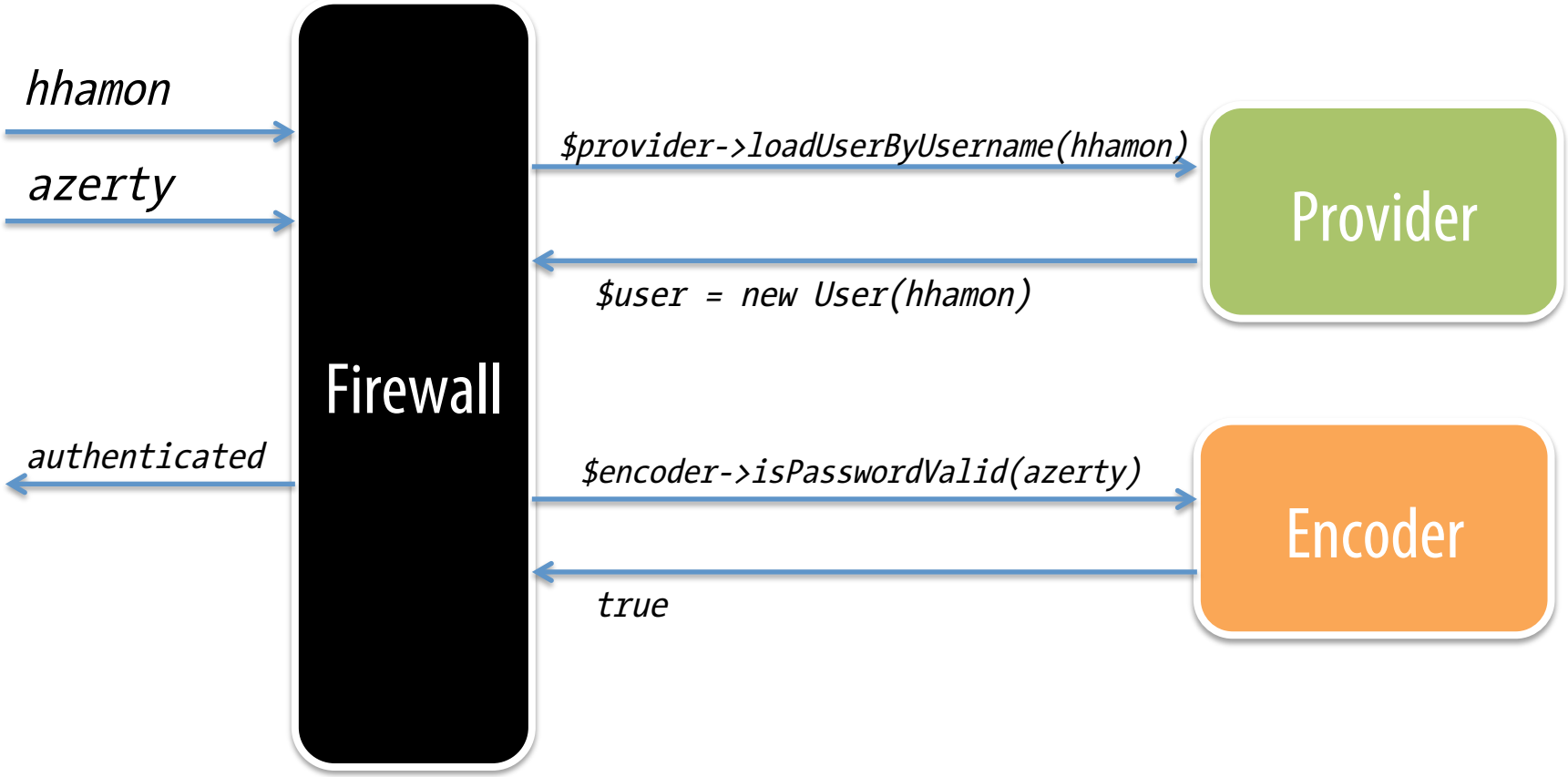
Security firewalls

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

Supported authentication firewalls

Method	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



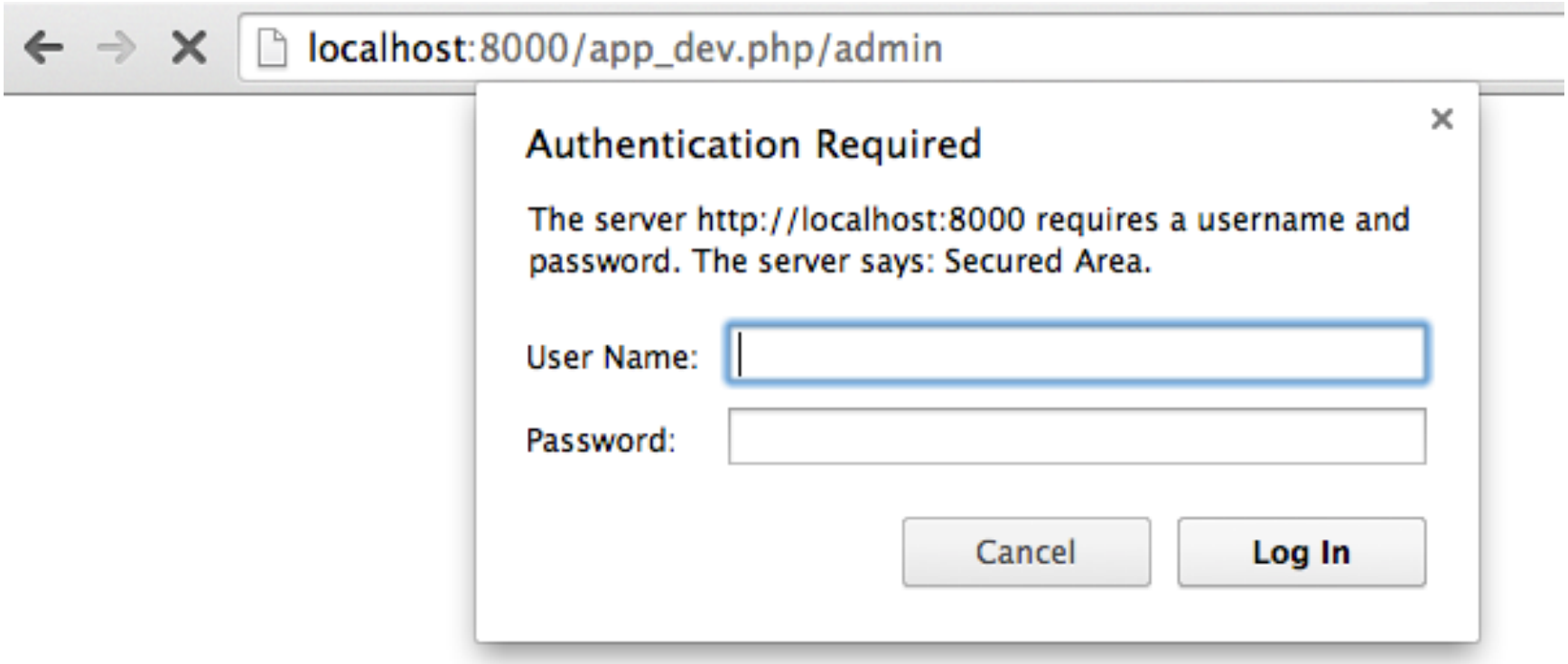
HTTP authentication

Configuring an HTTP Authentication

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

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

Authenticating with HTTP Basic

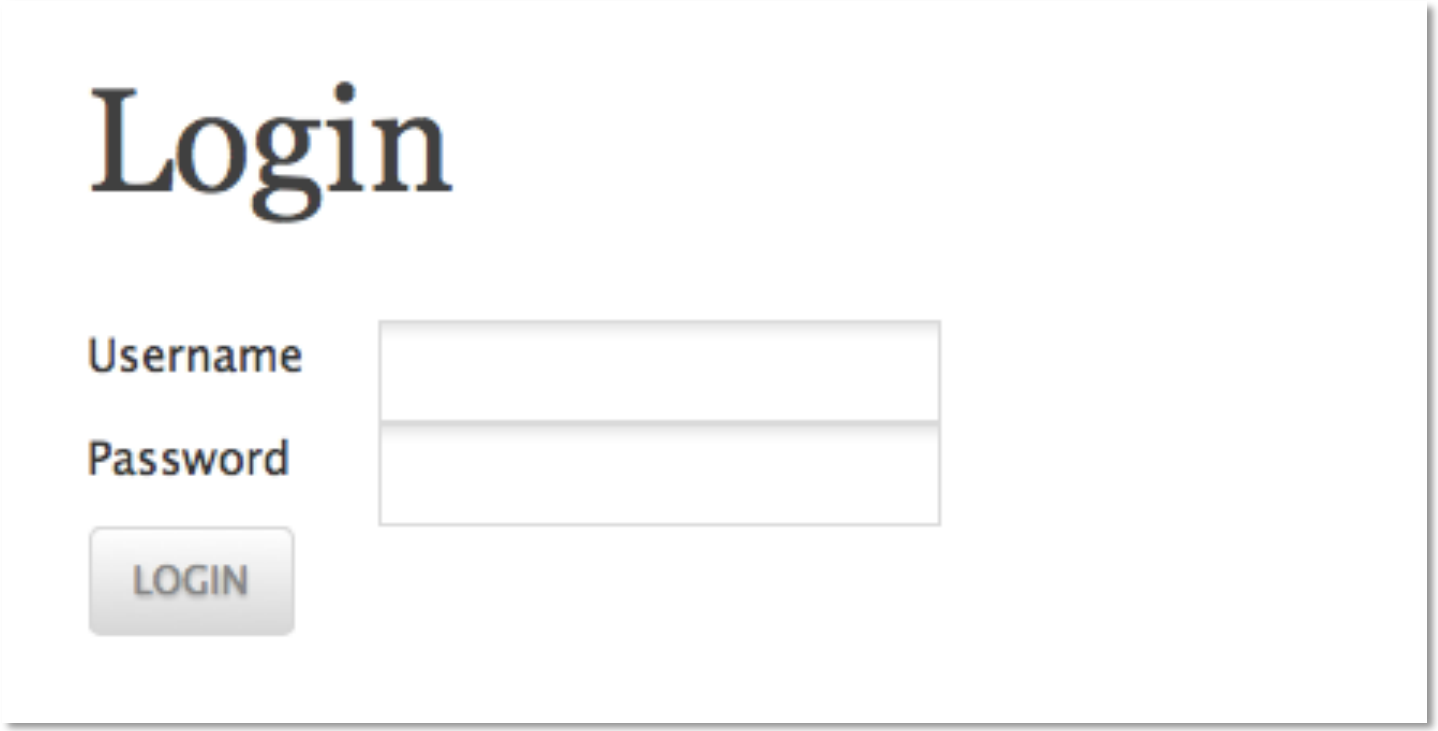


Login form authentication

Authenticating with a login form

```yaml
# app/config/security.yml
security:
    # ...
    firewalls:
        admin:
            provider:                   administrators
            pattern:                    "^/admin"
            form_login:
                check_path:             app_login_check
                login_path:             app_signin
                default_target_path:    app_admin
                always_use_default_target_path: true
```

The login form



Adding routes for authentication

```
# app/config/routing.yml
app_login_check:
    path:     /admin/auth
    methods:  POST

app_signin:
    path:     /login
    defaults: { _controller:
                "AppBundle:Security:login" }
    methods:  GET
```

The login controller

```
class SecurityController extends Controller
{
    public function loginAction()
    {
        $helper = $this->get('security.authentication_utils');

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

        return $this->render('login.html.twig', [
            'last_username' => $name,
            'error' => $error,
        ]);
    }
}
```

The login template

```
{% extends 'base.html.twig' %}

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

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

    <form action="{{ path("app_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 logout capability

```
# app/config/security.yml
security:
    # ...
    firewalls:
        admin:
            # ...
            logout:
                path:   app_signout
                target: app_signin
```

Adding route for logout

```
# app/config/routing.yml
app_signout:
    path:    /admin/logout
    methods: GET
```

Allowing anonymous authentication

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

Accessing the user

```
// from a Symfony controller
$user = $this->getUser();
```

```
{# from a Twig template #}
{{ app.user }}
```

Impersonating users

Allowing admin users to switch context

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

Allowing admin users to switch context

```yaml
# app/config/security.yml
security:
  providers:
    administrators:
      memory:
        users:
          superadmin:
            password: superman
            roles:
                - 'ROLE_ADMIN'
                - 'ROLE_ALLOWED_TO_SWITCH'
```

Switching to another security user

```
http://localhost:8000
  /admin
  ?_switch_user=hhamon
```

```
http://localhost:8000
  /admin
  ?_switch_user=_exit
```

Supported authentication tokens

Method	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

Controlling access

Access control rules secure some parts of the application according with permissions.

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

```
# app/config/security.yml
security:
  # ...
  access_control:
    -
      path: '^/admin'
      host: mydomain.foo
      ips: 192.0.0.0/8
      requires_channel: https
      roles: [ROLE_ADMIN, ROLE_MANAGER]
      allow_if: "request.headers.get('User-Agent')
matches /Firefox/i"
```

Granting or denying access to the user

```
public function editAction($id)
{
    // Option #1.
    if (!$this->isGranted('ROLE_ADMIN')) {
        throw new AccessDeniedException();
    }

    // Option #2.
    $this->denyAccessUnlessGranted('ROLE_ADMIN');
}
```

Granting or denying access to the user

```
use Sensio\Bundle\FrameworkExtraBundle
\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