Security

Powerful and complete Security for your apps!

By https://andreiabohner.org

Symfony 4.4

You can use:
- Security Component (Standalone)
- SecurityBundle (integrates Security Component on Symfony)

Security Component (Standalone)

Contains sub-components: Core, Http, Guard, Csrf

Install `$ composer require symfony/security-core`

Authentication

Who you are

(a token will be generated to represent you)

HTTP Security component uses listeners attached to kernel.request to create tokens.

Tokens

Create a token representing the user input

UsernamePasswordToken	username and password token
RememberMeToken	uses a browser cookie
SwitchUserToken	token representing a user who temporarily impersonates another one
AnonymousToken	represents an anonymous token

```
use Symfony\Component\Security\Core\Authentication\Token\UsernamePasswordToken;

$inputToken = new UsernamePasswordToken('john', 'myPassword987', 'default');
```

often used in traditional apps

input from the user

UsernamePasswordFormAuthenticationListener creates a UsernamePasswordToken based on the login form submit

---

User Providers

Load users from "some resource"

ChainUserProvider	calls several providers in a chain until one is able to handle the request
InMemoryUserProvider	simple non persistent user provider. Useful for testing, demonstration, prototyping, and for simple needs (a backend with a unique admin for instance) e.g.: fetch users from a PHP array
LdapUserProvider	user provider on top of LDAP
MissingUserProvider	dummy user provider used to throw proper exception when a firewall requires a user provider but none was defined

You can also create your own custom user provider

```
use Symfony\Component\Security\Core\User\InMemoryUserProvider;

$userProvider = new InMemoryUserProvider([
    'john' => [
        'password' => 'myPassword987',
        'roles' => ['POST_CREATE']
    ],
]);

$myUser = $userProvider->loadUserByUsername('john');
```

find the user

Password Encoders

Encode a plain text password and check if the password is valid

These encoders do not require a user-generated salt

- Argon2iPasswordEncoder
- BCryptPasswordEncoder
- NativePasswordEncoder
- SodiumPasswordEncoder

Hashes passwords using the best available encoder

- MigratingPasswordEncoder
- MessageDigestPasswordEncoder
- Pbkdf2PasswordEncoder
- PlaintextPasswordEncoder
- UserPasswordEncoder

```
use Symfony\Component\Security\Core\Encoder\EncoderFactory;
use Symfony\Component\Security\Core\Encoder\PlaintextPasswordEncoder;
use Symfony\Component\Security\Core\User\User;

$encoderFactory = new EncoderFactory([
    User::class => new PlaintextPasswordEncoder(),
]);

$encoderFactory->getEncoder(User::class)
    ->isPasswordValid($myUser->getPassword(), 'myPassword987', '');
```

get the encoder associated with this user (other users can use other encoders)

check if matches the user's password

Security

Symfony 4.4

Authenticate the Token: AuthenticationManager

AuthenticationManagerInterface is responsible for this

AuthenticationProviderManager - authentication manager based on authentication providers:

Authentication Providers

Transform an unauthenticated token (user input) into an authenticated token (security identity)

AnonymousAuthenticationProvider	validates AnonymousToken instances. Always returns a token representing an anonymous user
DaoAuthenticationProvider	uses a user provider (UserProviderInterface) to retrieve a user matching the input and then matches the password (using a password encoder UsernamePasswordToken).
LdapBindAuthenticationProvider	authenticates a user against an LDAP server
RememberMeAuthenticationProvider	authenticates a remember-me cookie
SimpleAuthenticationProvider	deprecated since Symfony 4.2, use Guard instead

most used

Instantiate the Authentication Manager

After:
- create a token representing the user input
- load the user from some User Provider
- encode & check the password
we can create the AuthenticationProviderManager

```
use Symfony\Component\Security\Core\Authentication\AuthenticationProviderManager;
use Symfony\Component\Security\Core\Authentication\Provider\DaoAuthenticationProvider;
use Symfony\Component\Security\Core\User\UserChecker;

$authenticationManager = new AuthenticationProviderManager([
    new DaoAuthenticationProvider(
        $userProvider,
        new UserChecker(),
        'default',
        $encoderFactory
    ),
]);
```

Check some "user flags" after the user is fetched from user provider (e.g. if the user is activated, ...)

The provider key (same provided when create the token). Used to make sure the token is from our app and to know which provider should handle the token

Authenticate the Token

Create an authenticated token

```
$authenticatedToken = $authenticationManager->authenticate($inputToken);
echo 'Hi '.$authenticatedToken->getUsername();
```

Authorization

what you are allowed to do (determine whether or not you have access to something)

Authorize Actions: AccessDecisionManager

The default implementation uses "Security voters" to decide whether the user is allowed to execute an action.
These voters are provided with an attribute (representing the action) and optionally some context (the subject of the action).

```
use Symfony\Component\Security\Core\Authorization\AccessDecisionManager;
use Symfony\Component\Security\Core\Authorization\Voter\RoleVoter;

$accessDecisionManager = new AccessDecisionManager([
    new RoleVoter('POST_'),
]);

$isSupervisor = $accessDecisionManager->decide(
    $authenticatedToken,
    ['POST_CREATE']
);
```

this voter checks if the User's getRoles() contains the provided attribute

POST_ is the prefix an attribute must have in order to be managed by this voter

uses the access decision manager to see if the authenticated token has the "POST_CREATE" role

Voters

The default Symfony voters don't validate an action, but validate the user's identity.

AuthenticatedVoter
ExpressionVoter
RoleVoter
RoleHierarchyVoter

By https://andreiabohner.org

Security

Symfony 4.4

Install

```
$ composer require symfony/security-bundle
```

Symfony (SecurityBundle)

Integrates the Security Component on Symfony apps

Authentication

The User Class

The easiest way to create the User class is to use the MakerBundle

```
$ php bin/console make:user

    The name of the security user class (e.g. User) [User]:
    > User

    Do you want to store user data in the database (via Doctrine)? (yes/no)[yes]:
    > yes

    Enter a property name that will be the unique "display" name for the user
    (e.g. email, username, uuid [email]
    > email

    Does this app need to hash/check user passwords? (yes/no) [yes]:
    > yes

    created: src/Entity/User.php
    created: src/Repository/UserRepository.php
    updated: src/Entity/User.php
    updated: config/packages/security.yaml
```

Call the class: User

e.g. config to store user info in the database

configured one User Provider in your security.yaml file under the providers key

```
use Symfony\Component\Security\Core\User\UserInterface;

/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 */
class User implements UserInterface
{
    // ...
}
```

The User class must implement UserInterface

Visual identifier that represents the user (isn't the username), could be an email for e.g. Only used to display who is currently logged in on the web debug toolbar.

```
public function getUsername(): string
{
    return (string) $this->email;
}
```

Enable the User Class as a User Provider

```
# config/packages/security.yaml
security:
    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
```

```
namespace Symfony\Component\Security\Core\User;
use Symfony\Component\Security\Core\Role\Role;

interface UserInterface
{
  /**
   * Returns the roles granted to the user.
   *
   *    public function getRoles()
   *    {
   *        return ['ROLE_USER'];
   *    }
   *
   * Alternatively, the roles might be stored on a ``roles`` property,
   * and populated in any number of different ways when the user object
   * is created.
   *
   * @return (Role|string)[] The user roles
   */
  public function getRoles();
  /**
   * Returns the password used to authenticate the user.
   *
   * This should be the encoded password. On authentication, a plain-text
   * password will be salted, encoded, and then compared to this value.
   *
   * @return string The password
   */
  public function getPassword();
  /**
   * Returns the salt that was originally used to encode the password.
   *
   * This can return null if the password was not encoded using a salt.
   *
   * @return string|null The salt
   */
  public function getSalt();
  /**
   * Returns the username used to authenticate the user.
   *
   * @return string The username
   */
  public function getUsername();
  /**
   * Removes sensitive data from the user.
   *
   * This is important if, at any given point, sensitive information like
   * the plain-text password is stored on this object.
   */
  public function eraseCredentials();
}
```

By https://andreiabohner.org

Security

Symfony 4.4

User Providers

Load users from some resource, reload User data from the session, and some other optional features, like remember me, and impersonation (switch_user). Configured under "providers" key in security.yml

Entity User Provider	loads users from database
LDAP User Provider	loads users from LDAP server
Memory User Provider	loads users from configuration file
Chain User Provider	merges two or more user providers into a new user provider

Entity User Provider

Common for traditional web apps. Users are stored in a database and the user provider uses Doctrine to retrieve them

```
providers:
    users:
        entity:
            class: 'App\Entity\User'
            property: 'username'
            # manager_name: 'customer'
```

the class of the entity that represents users

the property used to query by. (can only query from one field)

optional: if you're using multiple Doctrine entity managers, this option defines which one to use

Memory User Provider

Useful in app prototypes. Stores all user information in a configuration file, including their passwords

```
providers:
    backend_users:
        memory:
            users:
                john_admin: { password: '$2y$13$a...', roles: ['ROLE_ADMIN'] }
                jane_admin: { password: '$2y$13$C...', roles: ['ROLE_ADMIN', 'ROLE_SUPER_ADMIN'] }
```

Using a Custom Query to Load the User

e.g. you want to find a user by email or username

```
// src/Repository/UserRepository.php
namespace App\Repository;

use Doctrine\ORM\EntityRepository;
use Symfony\Bridge\Doctrine\Security\User\UserLoaderInterface;

class UserRepository extends EntityRepository implements UserLoaderInterface
{
  // ...

  public function loadUserByUsername($usernameOrEmail)
  {
    return $this->createQueryBuilder('u')
      ->where('u.username = :query OR u.email = :query')
      ->setParameter('query', $usernameOrEmail)
      ->getQuery()
      ->getOneOrNullResult();
  }
}
```

implement this interface

define the logic in this method

HEADS UP! remove the property key from the entity provider in security.yaml

LDAP User Provider

```
providers:
    my_ldap:
        ldap:
            service: Symfony\Component\Ldap\Ldap
            base_dn: dc=example,dc=com
            search_dn: "cn=read-only-admin,dc=example,dc=com"
            search_password: password
            default_roles: ROLE_USER
            uid_key: uid
```

```
$ composer require symfony/ldap
```

install

```
# config/services.yaml
services:
    Symfony\Component\Ldap\Ldap:
        arguments: ['@Symfony\Component\Ldap\Adapter\ExtLdap\Adapter']
    Symfony\Component\Ldap\Adapter\ExtLdap\Adapter:
        arguments:
            -   host: my-server
                port: 389
                encryption: tls
                options:
                    protocol_version: 3
                    referrals: false
```

configure the LDAP client in services.yaml

Chain User Provider

Combines two or more user providers (entity, memory, and LDAP) to create a new user provider

```
providers:
    backend_users:
        memory:
            # ...

    legacy_users:
        entity:
            # ...

    users:
        entity:
            # ...

    all_users:
        chain:
            providers: ['legacy_users', 'users', 'backend']
```

How Users are Refreshed from Session

End of every request	User object is serialized to the session
Beginning of the next request	User object it's deserialized & passed to the user provider to "refresh" it (e.g. Doctrine queries the DB for a fresh user). Then, the original User object from the session and the refreshed User object are "compared" to see if they are "equal". If any of these are different, your user will be logged out.

By default, the core AbstractToken class compares the return values of the: getPassword() getSalt() getUsername()

Security

Symfony 4.4

Custom User Provider

if you're loading users from a custom location (e.g. legacy database connection), you'll need to create a custom user provider

```php
// src/Security/UserProvider.php
namespace App\Security;

use Symfony\Component\Security\Core\Exception\UnsupportedUserException;
use Symfony\Component\Security\Core\Exception\UsernameNotFoundException;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\User\UserProviderInterface;

class UserProvider implements UserProviderInterface
{
    /**
     * Symfony calls this method if you use features like switch_user
     * or remember_me.
     *
     * If you're not using these features, you don’t need to implement
     * this method.
     *
     * @return UserInterface
     *
     * @throws UsernameNotFoundException if the user is not found
     */
    public function loadUserByUsername($username)
    {
        // Load a User object from your data source or throw UsernameNotFoundException.
        // The $username argument may not actually be a username:
        // it is whatever value is being returned by the getUsername() method in your User class.
        throw new \Exception('TODO: fill in loadUserByUsername() inside '.__FILE__);
    }

    /**
     * Refreshes the user after being reloaded from the session.
     *
     * When a user is logged in, at the beginning of each request, the
     * User object is loaded from the session and then this method is
     * called. Your job is to make sure the user's data is still fresh by,
     * for example, re-querying for fresh User data.
     *
     * If your firewall is "stateless: true" (for a pure API), this method is not called.
     *
     * @return UserInterface
     */
    public function refreshUser(UserInterface $user)
    {
        if (!$user instanceof User) {
            throw new UnsupportedUserException(sprintf('Invalid user class "%s".', get_class($user)));
        }

        // Return a User object after sure its data is "fresh" or throw a UsernameNotFoundException if user no longer exists
        throw new \Exception('TODO: fill in refreshUser() inside '.__FILE__);
    }

    public function supportsClass($class)
    {
        return User::class === $class;
    }
}
```

tells Symfony to use this provider for this User class

Enable the Custom User Provider

```yaml
# config/packages/security.yaml
security:
    providers:
        your_custom_user_provider:
            id: App\Security\UserProvider
```

the name of your user provider can be anything

Security

Symfony 4.4

Encoding Passwords

you can control how passwords are encoded in security.yaml

Defines the algorithm used to encode passwords.

If your app defines more than one user class, each of them can define its own encoding algorithm.

```
security:
    # ...
    encoders:
        App\Entity\User:
            algorithm: bcrypt
            cost: 12
```

your user class name

bcrypt or sodium are recommended. sodium is more secure, but requires PHP 7.2 or the Sodium extension

```
security:
    # ...
    encoders:
        App\Entity\User:
            algorithm: auto
```

recommended: will use the best algorithm available on your system

Use the UserPasswordEncoderInterface service to encode and check passwords. E.g. using fixtures:

```
// src/DataFixtures/UserFixtures.php

use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class UserFixtures extends Fixture
{
    private $passwordEncoder;

    public function __construct(UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->passwordEncoder = $passwordEncoder;
    }

    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setPassword($this->passwordEncoder->encodePassword(
            $user,
            'the_new_password'
        ));
    }
}
```

Manually Encode a Password

```
$ php bin/console security:encode-password
```

Authenticating Users

Instead of building a route & controller to handle login, you'll activate an authentication provider: some code that runs automatically before your controller is called.

Authentication Providers

At the beginning of every request, Symfony calls a set of "authentication listeners", or "authenticators"

```
form_login
http_basic
LDAP via HTTP Basic or Form Login
json_login
X.509 Client Certificate Authentication (x509)
REMOTE_USER Based Authentication (remote_user)
simple_form
simple_pre_auth
Guard Authenticator
```

recommended: allows you to control every part of the authentication process

If your application logs users in via a third-party service such as Google, Facebook or Twitter (social login), check out the HWIOAuthBundle community bundle.

Comparing Users Manually with EquatableInterface

If you need more control over the "compare users" process, make your User class implement EquatableInterface. Then, your isEqualTo() method will be called when comparing users.

Security

Symfony 4.4

Guard Authentication Provider

Create a Login Form Authenticator

```
$ php bin/console make:auth
```

create an authenticator

class used when you choose "Login form authenticator" on make:auth command

You can use AbstractGuardAuthenticator instead to create an API authenticator

```
namespace App\Security;
...
class LoginFormAuthenticator extends AbtractFormLoginAuthenticator
{
    public function __construct(UserRepository $userRepository,
       RouterInterface $router, CsrfTokenManagerInterface $csrfTokenManager,
       UserPasswordEncoderInterface $passwordEncoder)
    {
        ...
        $this->csrfTokenManager = $csrfTokenManager;
    }

    public function supports(Request $request)
    {
        // do your work when we're POSTing to the login page
        return $request->attributes->get('_route') === 'login'
         && $request->isMethod('POST');
    }

    public function getCredentials(Request $request)
    {
        return [
            'email' => $request->request->get('email'),
            'password' => $request->request->get('password'),
            'csrf_token' => $request->request->get('_csrf_token'),
        ];
    }

    public function getUser($credentials, UserProviderInterface $userProvider)
    {
        $token = new CsrfToken('authenticate', $credentials['csrf_token']);
        if (!$this->csrfTokenManager->isTokenValid($token)) {
            throw new InvalidCsrfTokenException();
        }
        return $this->userRepository->findOneBy(['email' => $credentials['email']]);
    }

    public function checkCredentials($credentials, UserInterface $user)
    {
        return $this->passwordEncoder->isPasswordValid($user, $credentials['password']);
    }

    public function onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
    {
        if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {
            return new RedirectResponse($targetPath);
        }
        return new RedirectResponse($this->router->generate('homepage'));
    }

    protected function getLoginUrl()
    {
        return $this->router->generate('login');
    }
}
```

if return:
false - nothing else happens. It doesn't call any other methods on the authenticator
true - call getCredentials()

read the authentication credentials of the request and return them.
Call getUser() and pass this array back to us as the first $credentials argument:

Use the $credentials to return a User object, or null if the user isn't found.

if return:
null - the authentication process stop, and the user will see an error.
User object - calls checkCredentials(), and passes to it the same $credentials and User object

check to see if the user's password is correct, or any other security checks.

if return:
false - authentication would fail and the user see an "Invalid Credentials" message.
true - authentication is successful, calls onAuthenticationSuccess()

where to redirect after a successful login

if there is a referer, redirect to it, if not, to homepage

if return:
- Response object: will be immediately sent back to the user
- nothing: the request would continue to the controller

on failure, the authenticator class calls getLoginUrl() and try to redirect here

Enable the Authenticator

```
# config/packages/security.yaml
firewalls:
    main:
        guard:
            authenticators:
                - App\Security\LoginFormAuthenticator
```

When activated, at the beginning of every request, the supports() method of the authenticator will be called

CSRF Protection

templates/security/login.html.twig

```
...
    <input type="hidden" name="_csrf_token"
        value="{{ csrf_token('authenticate') }}">
...
```

Security

Symfony 4.4

Guard Authentication Provider

Guard Authenticator Methods

```
supports(Request $request)
getCredentials(Request $request)
getUser($credentials, UserProviderInterface $userProvider)
checkCredentials($credentials, UserInterface $user)
onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
onAuthenticationFailure(Request $request, AuthenticationException $exception)
start(Request $request, AuthenticationException $authException = null)
supportsRememberMe()
```

you don't need to handle these 3 methods when using AbtractFormLoginAuthenticator. They are handled automatically

Login and Logout Methods

```
// src/Controller/SecurityController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends AbstractController
{
    /**
     * @Route("/login", name="login")
     */
    public function login(AuthenticationUtils $authenticationUtils)
    {

        $error = $authenticationUtils->getLastAuthenticationError();

        $lastUsername = $authenticationUtils->getLastUsername();
        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error'         => $error,
        ]);
    }

    /**
     * @Route("/logout", name="logout")
     */
    public function logout()
    {
    }
}
```

the route name compared in supports method of LoginFormAuthenticator class

get the login error if there is one

last username entered by the user

just write the method and add the path defined in security.yaml Symfony will automatically log the user out and then redirect them

Control What Happens After Logout

```
# config/packages/security.yaml
security:
    firewalls:
        main:
            logout:
                path: logout
                success_handler: logout_success_handler
```

point it to a service id of a class that implements LogoutSuccessHandlerInterface

Display Login Error Messages in Templates

templates/security/login.html.twig

```
{% if error %}
{{ error.messageKey|trans(error.messageData, 'security') }}
{% endif %}
```

Security

Symfony 4.4

Remember Me

This is the special name that Symfony uses

```
<input type="checkbox" name="_remember_me"> Remember me
```

Enable it

```
# config/packages/security.yaml
security:
    firewalls:
        main:
            remember_me:
                secret:   '%kernel.secret%'
                lifetime: 2592000 # 30 days in seconds
```

Impersonating a User

You can go to any URL and add ?_switch_user= and the user identifier (e.g. email) of an user that you want to impersonate.

```
http://example.com/somewhere?_switch_user=john@example.com
```

Enable it

```
security:
    firewalls:
        main:
            switch_user: true
```

Requires you to have the ROLE_ALLOWED_TO_SWITCH

```
security:
    role_hierarchy:
        ROLE_ADMIN: [ROLE_ALLOWED_TO_SWITCH]
```

Switch Back to the Original User

```
?_switch_user=_exit
```

```
http://example.com/somewhere?_switch_user=_exit
```

Find the Original User

```
$token = $this->security->getToken();

if ($token instanceof SwitchUserToken) {
    $impersonatorUser = $token->getOriginalToken()->getUser();
}
```

Knowing when Impersonation is Active

When we are switched to another user, Symfony gives us a special role called ROLE_PREVIOUS_ADMIN

```
{% if is_granted('ROLE_PREVIOUS_ADMIN') %}
    <a href="{{ path('homepage', {'_switch_user': '_exit'}) }}">Exit</a>
{% endif %}
```

Custom User Checker

if you need additional checks before and after user authentication

```
namespace App\Security;

use App\Security\User as AppUser;
use Symfony\Component\Security\Core\Exception\AccountExpiredException;
use App\Exception\AccountDeletedException;
use Symfony\Component\Security\Core\User\UserCheckerInterface;
use Symfony\Component\Security\Core\User\UserInterface;

class UserChecker implements UserCheckerInterface
{
    public function checkPreAuth(UserInterface $user)
    {
        if (!$user instanceof AppUser) {
            return;
        }

        if ($user->isDeleted()) {
            throw new AccountDeletedException();
        }
    }

    public function checkPostAuth(UserInterface $user)
    {
        if (!$user instanceof AppUser) {
            return;
        }

        if ($user->isExpired()) {
            throw new AccountExpiredException('...');
        }
    }
}
```

must implement UserCheckerInterface

user is deleted, show a generic Account Not Found message

user account is expired, the user may be notified

Enable it

```
# config/packages/security.yaml
security:
    firewalls:
        main:
            pattern: ^/
            user_checker: App\Security\UserChecker
```

defined per firewall

Security

Symfony 4.4

Authorization

Decide if a user can access some resource
This decision will be made by an instance of AccessDecisionManagerInterface

The Authorization Process Consists of:

1. Add roles: user receives a specific set of roles when logging in (e.g. ROLE_ADMIN)
2. Check permissions: a resource (e.g. URL, controller) requires a specific role (like ROLE_ADMIN) to be accessed

1. ROLES (define what the user can access)

When a user logs in, the getRoles() method on your User object is called to determine which roles the user has

Are strings used to grant access to users (e.g. "edit a blog post", "create an invoice"). You can freely choose those strings.
The only requirement is that they must start with `ROLE_` (e.g. `ROLE_POST_EDIT`, `ROLE_INVOICE_CREATE`).

Standard ROLES

you have to return at least one role (e.g. ROLE_USER) for the user

`ROLE_USER`	Add to all logged users	`ROLE_PREVIOUS_ADMIN`	Added when we are switched to another user
`ROLE_ADMIN`		`ROLE_ALLOWED_TO_SWITCH`	Allow switch to another user
`ROLE_SUPER_ADMIN`		`ROLE_YOUR_DEFINED_NAME`	

Special "ROLES"

you can use these anywhere roles are used: like access_control, controller or in Twig.
you can use these roles to check if a user is logged in

`IS_AUTHENTICATED_REMEMBERED`	All logged in users have this. Even if you don't use the remember me functionality, you can use this to check if the user is logged in
`IS_AUTHENTICATED_FULLY`	Users who are logged in only because of a "remember me" have IS_AUTHENTICATED_REMEMBERED but not have IS_AUTHENTICATED_FULLY
`IS_AUTHENTICATED_ANONYMOUSLY`	All users (even anonymous ones) have this

---

2. Checking Permissions (handle authorization)

Calls the "voter" system
Symfony takes the responses from all voters and makes the final decision (allow or deny access to the resource) according to the strategy defined (affirmative, consensus or unanimous

- for protecting broad URL patterns, use `access_control` in `security.yaml`
- whenever possible, use the `@Security` annotation in your controller
- check security directly on the `security.authorization_checker` service (isGranted) for complex situations
- define a custom security voter to implement fine-grained restrictions

Checking Permissions in the Controller

You can use:
- annotations (@Security or @IsGranted)
- methods (denyAccessUnlessGranted() or isGranted())

Using @Security Annotation

```
use App\Entity\Post;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
 * @Security("is_granted('ROLE_ADMIN')")
 */
public function new()
{
}

/**
 * @Security("user.getEmail() == post.getAuthorEmail()")
 */
public function edit(Post $post)
{
}
```

Using @IsGranted Annotation

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;

/**
  * @IsGranted("ROLE_ADMIN")
  */
class AdminController extends AbstractController
{
     /**
      * @IsGranted("ROLE_ADMIN")
      */
    public function adminDashboard()
    {
    }
}
```

require ROLE_ADMIN for *every* method in this class

require ROLE_ADMIN for only this method

Security

Symfony 4.4

Using isGranted() and denyAccessUnlesssGranted() Methods

```
if (!$post->isAuthor($this->getUser())) {
   $this->denyAccessUnlessGranted('edit', $post);
}
```

Equivalent code without using the "denyAccessUnlessGranted()" shortcut:

```
use Symfony\Component\Security\Core\Exception\AccessDeniedException;
use Symfony\Component\Security\Core\Authorization\AuthorizationCheckerInterface
 ...

public function __construct(AuthorizationCheckerInterface $authorizationChecker)
{
     $this->authorizationChecker = $authorizationChecker;
}
...

if (!$this->authorizationChecker->isGranted('edit', $post)) {
    throw $this->createAccessDeniedException();
}
```

If access is not granted, a AccessDeniedException is thrown:

- not logged: redirect to the login page
- logged in: show the 403 access denied page

```
$this->denyAccessUnlessGranted('ROLE_ADMIN');
$hasAccess = $this->isGranted('ROLE_ADMIN');
```

Checking Permissions in Templates (Twig)

```
{% if is_granted('ROLE_USER') %}
      ....
{% else %}
       <a href="{{ path('app_login') }}">Login</a>
{% endif %}
```

Get the User Who is Logged In

Controller

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

Template (Twig)

```
{{ app.user.firstName }}
```

Service

```
use Symfony\Component\Security\Core\Security;
class SomeService
{
    private $security;

    public function __construct(Security $security)
    {
        $this->security = $security;
    }

    public function someMethod(): string
    {
        $user = $this->security->getUser();
    }
}
```

Access Decision Manager

deciding whether or not a user is authorized to perform a certain action

Depends on multiple voters, and makes a final verdict based on all the votes (either positive, negative or neutral) it has received. It recognizes several strategies:

affirmative (default)	grant access as soon as there is one voter granting access
consensus	grant access if there are more voters granting access than there are denying
unanimous	only grant access if none of the voters has denied access

```
use Symfony\Component\Security\Core\Authorization\AccessDecisionManager;

$voters = [...];  // instances of Symfony\Component\Security\Core\Authorization\Voter\VoterInterface

$strategy = ...;  // one of "affirmative", "consensus", "unanimous"

$allowIfAllAbstainDecisions = ...;  // whether or not to grant access when all voters abstain

$allowIfEqualGrantedDeniedDecisions = ...;  // whether or not to grant access when there is no majority (only to the "consensus" strategy)

$accessDecisionManager = new AccessDecisionManager(
    $voters,
    $strategy,
    $allowIfAllAbstainDecisions,
    $allowIfEqualGrantedDeniedDecisions
);
```

Change the Default Strategy

```
# config/packages/security.yaml
security:
    access_decision_manager:
        strategy: unanimous
        allow_if_all_abstain: false
```

Security

Symfony 4.4

Are the most granular way of checking permissions → Voters

When your security logic is complex use custom voters → Creating a Custom Voter

```php
namespace App\Security;

use App\Entity\Post;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Authorization\AccessDecisionManagerInterface;
use Symfony\Component\Security\Core\Authorization\Voter\Voter;
use Symfony\Component\Security\Core\User\UserInterface;

class PostVoter extends Voter // or implement VoterInterface
{
    const CREATE = 'create';
    const EDIT   = 'edit';

    private $decisionManager;
    private $security;

    public function __construct(AccessDecisionManagerInterface $decisionManager,
                                Security $security)
    {
        $this->decisionManager = $decisionManager;
        $this->security = $security;
    }

    protected function supports($attribute, $subject)
    {
        if (!in_array($attribute, [self::CREATE, self::EDIT])) {
            return false;
        }

        if (!$subject instanceof Post) {
            return false;
        }

        return true;
    }

    protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
    {
        $user = $token->getUser();

        if (!$user instanceof UserInterface) {
            return false;
        }

        if ($this->security->isGranted('ROLE_SUPER_ADMIN')) {
            return true;
        }

        /** @var Post $post */
        $post = $subject;

        switch ($attribute) {
            case self::CREATE:
                if ($this->decisionManager->decide($token, ['ROLE_ADMIN'])) {
                    return true;
                }
                break;

            case self::EDIT:
                if ($user->getEmail() === $post->getAuthorEmail()) {
                    return true;
                }
                break;
        }

        return false;
    }
}
```

When isGranted() or denyAccessUnlessGranted() is called, the first argument is passed here as $attribute (e.g. ROLE_USER, edit) and the second argument (if any) is passed as $subject (e.g. null, a Post object)

If return false this voter is done: some other voter should process this

If return true, voteOnAttribute() will be called

The $token can be used to find the current user object (if any)

the user must be logged in; if not, deny access

Checking for Roles inside a Voter

you know $subject is a Post object, thanks to supports method

if the user is an admin, allow them to create new posts

if the user is the author of the post, allow them to edit the posts

return:
true - to allow access
false - to deny access

Using the Custom Voter

you can use the voter with the @Security annotation:

```php
/**
 * @Security("is_granted('edit', post)")
 */
public function edit(Post $post)
{
    // ...
}
```

You can also use this directly with the security.authorization_checker service or via the even easier shortcut in a controller:

```php
/**
 * @Route("/{id}/edit", name="admin_post_edit")
 */
public function edit($id)
{
    $post = ...; // query for the post

    $this->denyAccessUnlessGranted('edit', $post);


}
```

By https://andreiabohner.org

Security

Symfony 4.4

Where the security system is configured

SecurityBundle Configuration (security.yaml)

```yaml
# config/packages/security.yaml
security:
    access_denied_url: null # where user is redirected after a 403 HTTP error (unless there is a custom access deny handler)

    always_authenticate_before_granting: false # if true: user is asked to authenticate before each call to the isGranted() in services, controllers, or templates

    erase_credentials: true # if true: eraseCredentials() method of the user object is called after authentication

    hide_user_not_found: true # if true: when a user isn't found a generic BadCredentialsException exception is thrown w/ msg "Bad credentials" if false: UsernameNotFoundException exception is thrown and includes the given not found username

    session_fixation_strategy: migrate # protection against session fixation. Possible values: NONE: session isn't changed, MIGRATE: session id is updated, attributes are kept, INVALIDATE: session id is updated, attributes are lost
```

User Providers

```yaml
    providers: # user providers
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
```

```yaml
providers:
    users:
        entity:
            class: 'App\Entity\User'
            property: 'username'
            # manager_name: 'customer'
    my_ldap:
        ldap:
            service: Symfony\Component\Ldap\Ldap
            base_dn: dc=example,dc=com
            search_dn: "cn=read-only-admin,dc=example,dc=com"
            ...
    backend_users:
        memory:
            users:
               user:  {password: userpass, roles: ['ROLE_USER']}
               admin: {password: adminpass, roles: ['ROLE_ADMIN']}
    custom_user_provider:
        id: App\Security\UserProvider
    all_users:
        chain:
            providers: ['my_ldap', 'users', 'backend']
```

Password Encoders

```yaml
    encoders: # password encoders
        App\Entity\User:
            algorithm: auto # use the best possible algorithm available on your system

        App\Entity\User: 'bcrypt' # bcrypt encoder with default options

        App\Entity\User:
            algorithm: 'bcrypt' # bcrypt encoder with custom options
            cost:      15

        App\Entity\User: 'sodium' # sodium encoder with default options

        App\Entity\User:
            algorithm:   'sodium' # sodium encoder with custom options
            memory_cost:  16384 # Amount in KiB (16384=16 MiB)
            time_cost:    2 # Number of iterations
            threads:      4 # Number of parallel threads

        App\Entity\User:
            algorithm: argon2i # argon2i encoder with custom options
            memory_cost: 256
            time_cost: 1
            threads: 2

        App\Entity\User: 'sha512' # PBKDF2 encoder using SHA512 hashing with default options

        app_encoder: # custom named encoder: create your own password encoders as services
            id: 'App\Security\Encoder\MyCustomPasswordEncoder'

    extra_secure:
        algorithm: sodium
        memory_cost:  16384
        time_cost:    2
        threads:      4
```

Different Password Encoder for Each User

```php
// src/Acme/UserBundle/Entity/User.php
namespace Acme\UserBundle\Entity;

use Symfony\Component\Security\Core\Encoder\EncoderAwareInterface;
use Symfony\Component\Security\Core\User\UserInterface;

class User implements UserInterface, EncoderAwareInterface
{
    public function getEncoderName()
    {
        if ($this->isAdmin()) {
            return 'extra_secure'; // use the 'extra_secure' encoder only for admin users
        }

        return null; // use the default encoder
    }
}
```

By https://andreiabohner.org

Security

Symfony 4.4

Authentication

Firewalls are listeners of the HTTP component that defines the authentication mechanism used for each URL (or URL pattern) of your app

all firewalls are one AuthenticationProviderManager (and thus, one security system)

```yaml
firewalls:
    main:                                   # name of the firewall (can be chosen freely)
        switch_user: true                   # impersonating users can be done by activating the switch_user firewall listener
        # switch_user:
        #     role: ROLE_ADMIN              # allow change the ROLE
        #     parameter: _change_user       # and query string used

        # AnonymousAuthenticationProvider
        anonymous: true                     # allow anonymous requests so users can access public pages

        # Use UsernamePasswordToken & DaoAuthenticationProvider
        form_login: true

        logout:
            path:   app_logout
            target: app_any_route           # where to redirect after logout
            success_handler: logout_success_handler

        remember_me:
            secret: '%kernel.secret%'
            lifetime: 604800 # 1 week in seconds   # default: one year
            path:     /
            domain: null
            secure: false
            httponly: true
            samesite: null                  # if set to 'strict', the cookie will not be sent with cross-site requests
            remember_me_parameter: _remember_me
            catch_exceptions: false
            token_provider: token_provider_id
            #always_remember_me: true       # always enable remember me

        stateless: false

        user_checker: App\Security\UserChecker   # enable custom user checker
```

Authentication Providers

```yaml
main:
    # ...
        x509:                               # your web server is doing all the authentication process itself
            provider: your_user_provider
        remote_user:
            provider: your_user_provider
        simple_preauth:
            # ...
        guard:
            authenticators:                 # multiple guard authenticators using shared (one) entry point
                - App\Security\LoginFormAuthenticator
                - App\Security\FacebookConnectAuthenticator
            entry_point: App\Security\LoginFormAuthenticator

        form_login:                         # handles a login form POST automatically
            login_path: /login
            check_path: /login_check
            csrf_token_generator: security.csrf.token_manager
            csrf_parameter: _csrf_token
            csrf_token_id: a_private_string
            default_target_path: /after_login_route_name
            always_use_default_target_path: false
            use_referer: false
            failure_path: login_failure_route_name
            target_path_parameter: _target_path
            failure_path_parameter: back_to
            username_parameter: _username   # name of the username field
            password_parameter: _password   # name of the password field
            post_only: true
            use_forward: false              # if true: user will be forwarded to the login form instead of redirected
        form_login_ldap:
            service: Symfony\Component\Ldap\Ldap
            dn_string: 'uid={username},dc=example,dc=com'
        json_login:
            check_path:    login
            username_path: security.credentials.login
            password_path: security.credentials.password
        simple_form:
            # ...
        http_basic:                         # asks credentials (username & password) using a dialog in the browser. You cannot use logout with http_basic
            realm: Secured Area
        http_basic_ldap:
            service: Symfony\Component\Ldap\Ldap
            dn_string: 'uid={username},dc=example,dc=com'
        http_digest:
            # ...

api:                                        # name of the firewall
    pattern: ^/api/                         # multiple guard authenticators using separate entry points (firewall)
    guard:
        authenticators:
            - App\Security\ApiTokenAuthenticator
```

Restrict Firewalls to a Request

```yaml
security:
    firewalls:
        secured_area:                       # name of the firewall
            pattern: ^/admin                # by path
            host: ^admin\.example\.com$     # by host
            methods: [GET, POST]            # by HTTP methods
            request_matcher: app.firewall.secured_area.request_matcher   # by service
```

'pattern' is a regexp matched against the request URL. If there's a match, authentication is triggered

Security

Symfony 4.4

Authorization

Only one path will be matched per request: Symfony starts at the top of the list and as soon as it finds one access control that matches the URL, it uses that and stops.

The order of paths is important!

Each access_control can also match on IP address, hostname and HTTP methods. It can also be used to redirect a user to the https version of a URL pattern

Matching Options can be:
- path
- ip or ips (netmasks are supported)
- port
- host
- methods

```
access_control:
    - { path: ^/login$, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/internal, roles: IS_AUTHENTICATED_ANONYMOUSLY, ips: [127.0.0.1, ::1, 192.168.0.1/24] }
    - { path: ^/internal, roles: ROLE_NO_ACCESS }
    -
        path: ^/_internal/secure
        allow_if: "'127.0.0.1' == request.getClientIp() or is_granted('ROLE_ADMIN')"
    # matches /admin/users/*
    - { path: ^/admin/users, roles: ROLE_SUPER_ADMIN }
    # matches /admin/* except for anything matching the above rule
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
    - { path: ^/admin, roles: ROLE_USER_IP, ip: 127.0.0.1 }
    - { path: ^/admin, roles: ROLE_USER_PORT, ip: 127.0.0.1, port: 8080 }
    - { path: ^/admin, roles: ROLE_USER_HOST, host: symfony\.com$ }
    - { path: ^/admin, roles: ROLE_USER_METHOD, methods: [POST, PUT] }
    - { path: ^/admin, roles: ROLE_USER }
    - { path: ^/cart/checkout, roles: IS_AUTHENTICATED_ANONYMOUSLY, requires_channel: https }
```

'ips' option supports IP addresses and subnet masks

using an expression

force redirect to HTTPs

Instead of giving many roles to each user, you can define role inheritance rules by creating a role hierarchy

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


access_decision_manager:
    strategy: unanimous
    allow_if_all_abstain: false
```

change the default access decision strategy (decide whether or not a user is authorized to perform a certain action using voters)

Console

```
# displays the default config values defined by Symfony
$ php bin/console config:dump-reference security

# displays the actual config values used by your app
$ php bin/console debug:config security
```

Standard Voters

AuthenticatedVoter	checks if the token is fully authenticated, anonymous, ... votes if IS_AUTHENTICATED_FULLY, IS_AUTHENTICATED_REMEMBERED, or IS_AUTHENTICATED_ANONYMOUSLY is present.
ExpressionVoter	votes based on the evaluation of an expression created with the ExpressionLanguage component
RoleVoter	votes if any attribute starts with a given prefix. (supports attributes starting with ROLE_ and grants access to the user when the required ROLE_* attributes can all be found in the array of roles returned by the token's getRoleNames() method)
RoleHierarchyVoter	understands hierarchies in roles (e.g. "admin is a user"). Extends RoleVoter and uses a RoleHierarchy to determine the roles granted to the user before voting