One of my current projects is using JWT Authentication for an API built on Symfony (shocker, I know). One of the things I wanted to build into this was a limiter on the number of "login" attempts for the API that would lock someone out for *x minutes* after *n* failed attempts; mirroring typical traditional form login behavior.

This was accomplished with the following components:

- Separate non-entity-driven database table
- Parameters for defining the *number of minutes to lock* and the *number of attempts* before triggering a a lock.
- A simple handler to check our table and do some simple maintenance and querying.

Separate non-entity-driven database table

The database table to track login attempts needed to be separate from a Doctrine (the ORM in this case) entity because a failed attempt naturally did not map to a real User. Similarly, we didn't want to track ip_addr of logged in users because we are not Vizio. So, that being said, we created a very simple database table to track login attempts by ip_addr:

```
<?php

namespace Application\Migrations;

use Doctrine\DBAL\Migrations\AbstractMigration;
use Doctrine\DBAL\Schema\Schema;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
class LoginAttempts extends AbstractMigration
{
    /**
```

```
     * @param Schema $schema
     */
    public function up(Schema $schema)
    {
        // this up() migration is auto-generated, please modify it to
your needs
$this->abortIf($this->connection->getDatabasePlatform()->getName() !==
'mysql', 'Migration can only be executed safely on \'mysql\'.');
        $this->addSql('CREATE TABLE login_attempts (ip_addr
VARCHAR(500) DEFAULT NULL COLLATE latin1_swedish_ci, user_agent
VARCHAR(500) DEFAULT NULL COLLATE latin1_swedish_ci, login_timestamp
DATETIME DEFAULT NULL) DEFAULT CHARACTER SET utf8 COLLATE
utf8_unicode_ci ENGINE = InnoDB');

    }

    /**
     * @param Schema $schema
     */
    public function down(Schema $schema)
    {
        // this down() migration is auto-generated, please modify it
to your needs
$this->abortIf($this->connection->getDatabasePlatform()->getName() !==
'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE login_attempts');
    }
}
```

Parameters

Again in this case, we wanted to parameterize the values we used to determine the number of attempts, and the number of minutes a user was locked out from attempting to login. With Symfony this is a simple addition to the parameters file.

```
#app/config/parameters.yml
parameters:
    login_attempt_minutes: 1
    max_login_attempts: 3
    timezone: "America/Los_Angeles" # This should already exist in
your app, but just in case.
```

It's important to note that timezone is very important here for comparing dates (later), so make sure when you define this (or if you ever change it) that you clear out the `login_attempts` table as well.

LoginAttemptHandler

Most importantly, we had to build a simple handler to manage our queries and maintenance actions. Ultimately, this is where the magic happens but the most important part is still last where we handle the actual logic in our Authentication controller (see below). However, this handler does the heavy lifting, and does have some room for improvement, particularly with Unit Testing.

```
<?php namespace MyApp\Bundle\MyAppBundle\Services;

use Doctrine\ORM\EntityManager;
use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Symfony\Component\HttpFoundation\Request;

class LoginAttemptHandler implements ContainerAwareInterface
{
```

```
    protected $container;

    protected $em;
    protected $login_attempt_minutes;
    protected $max_login_attempts;

    /**
     * Constructor.
     *
     * @param \Doctrine\ORM\EntityManager $em
     */
    public function __construct(EntityManager $em,
$login_attempt_minutes, $max_login_attempts)
    {
        $this->em = $em;
        $this->max_login_attempts = $max_login_attempts;
        $this->login_attempt_minutes = $login_attempt_minutes;
    }

    /**
     * @param ContainerInterface $container
     */
    public function setContainer(ContainerInterface $container = null)
    {
        $this->container = $container;
    }

    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @throws \Exception
```

```
     */
    public function log(Request $request)
    {
        try {

            $now = new \DateTime();
            $now->setTimezone(new
\DateTimeZone($this->container->getParameter('timezone')));

            $conn = $this->em->getConnection();
            $conn->insert('login_attempts', [
                'ip_addr' => $request->getClientIp(),
                'user_agent' => $request->headers->get('User-Agent'),
                'login_timestamp' => $now->format('Y-m-d H:i:s')
            ]);

        }
        catch(\Exception $e) {
            throw $e;
        }
    }

    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @throws \Exception
     */
    public function clear(Request $request)
    {
        try {
            $sql = "DELETE FROM login_attempts WHERE ip_addr =
```

```
:ip_addr";

            $params = array('ip_addr' => $request->getClientIp());
            $stmt   = $this->em->getConnection()->prepare($sql);
            $stmt->execute($params);
        }
        catch(\Exception $e) {
            throw $e;
        }
    }

    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return mixed
     * @throws \Exception
     */
    public function get(Request $request)
    {
        try {

            /**
             * If after three attempts you've still failed, you'll
have to wait
             * 15 minutes to attempt to login again..
             */
            $conn = $this->em->getConnection();
            $sql = 'SELECT * FROM login_attempts WHERE ip_addr =
:ip_addr';
            $params = array('ip_addr' => $request->getClientIp());
            $stmt = $conn->prepare($sql);
```

```
            $stmt->execute($params);
            $attempts = $stmt->fetchAll();

            return $attempts;

        }
        catch(\Exception $e) {
            throw $e;
        }
    }

    /**
     * @param array $attempts
     *
     * @return int|string
     * @throws \Exception
     */
    public function getLockoutMinutes(array $attempts = array())
    {
        try {

            $now = new \DateTime();
            $now ->setTimezone(new
\DateTimeZone($this->container->getParameter('timezone')));
            $minutes = 0;

            foreach($attempts as $attempt) {
                $timestamp = $attempt['login_timestamp'];
                $loginAttempt = new \DateTime($timestamp);
                $interval = $now->diff($loginAttempt);
                $minutes = $interval->format('%i');
```

```
            }

            return $minutes;

        }
        catch(\Exception $e) {
            throw $e;
        }
    }

    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return bool
     * @throws \Exception
     */
    public function lock(Request $request)
    {
        try {

            $attempts = $this->get($request);
            $minutes = $this->getLockoutMinutes($attempts);
            return (count($attempts) >= 3 && $minutes <=
$this->login_attempt_minutes) ? true : false;

        }
        catch(\Exception $e) {
            throw $e;
        }
    }
```

```
    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return bool
     * @throws \Exception
     */
    public function unlock(Request $request)
    {
        try {

            $attempts = $this->get($request);
            $minutes = $this->getLockoutMinutes($attempts);
            return (count($attempts) > $this->max_login_attempts &&
$minutes > $this->login_attempt_minutes) ? true : false;

        }
        catch(\Exception $e) {
            throw $e;
        }
    }

}
```

Define LoginAttemptHandler as a Service

Symfony needs to understand this handler is a service, so we can use it in our Controller (next).

```
# app/src/MyApp/Bundle/MyAppBundle/Resources/config/services.yml
services:

    myapp.login_attempts:
```

```
        class: MyApp\Bundle\MyAppBundle\Services\LoginAttemptHandler
        calls:
            - [setContainer, ['@service_container']]
        arguments: ['@doctrine.orm.entity_manager',
'%login_attempt_minutes%', '%max_login_attempts%']
```

Handle Login Attempts in our Authentication Controller

Now, every time there is a JWT Token request, we have a slight amount of extra overhead to verify login attempts in our Controller.

```
<?php namespace MyApp\Bundle\MyAppBundle\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\JsonResponse;

use Symfony\Component\Security\Core\Exception\AccessDeniedException;

class AuthController extends Controller
{

    /**
     * @param \Symfony\Component\HttpFoundation\Request $request
     *
     * @return \Symfony\Component\HttpFoundation\JsonResponse
     * @throws \Exception
     */
    public function tokenAuthenticateAction(Request $request)
    {
        $loginAttemptHandler =
$this->container->get('myapp.login_attempts');
```

```
        $username = $request->request->get('username');
        $password = $request->request->get('password');

        if($loginAttemptHandler->lock($request)) {
            throw new \Exception(sprintf('You have reached the maximum
number of login attempts. Please try again in %s minutes.',
$this->getParameter('login_attempt_minutes')));
        }
        elseif($loginAttemptHandler->unlock($request)) {
            $loginAttemptHandler->clear($request);
        }

        // fetch your User from the data storage with $username and
$password

        // if password is invalid, log it..
if(!$this->get('security.password_encoder')->isPasswordValid($user,
$password)) {
            $loginAttemptHandler->log($request);
            throw $this->createAccessDeniedException();
        }

        // fetch your $token from wherever you do so..
        //clear all login requests for this "user"
        $loginAttemptHandler->clear($request);

        // Return genereted tocken
        return new JsonResponse(['token' => $token]);
    }

}
```

Share this:

- LinkedIn
- Twitter
- Email
- Print