I recently built an app that required we update the authenticated user's lastOnlineDate every session so we can quickly track who has been online in the last interval of time. We set this interval as a parameter and used that to avoid hitting the database more than absolutely necessary to accomplish this.

To make it work, we needed to leverage onSecurityInteractiveLogin:

app/config/parameters.yml

```
parameters:
    activity_minutes: 10
    activity_days: 14
```

MyApp/EventListener/LoginListener.php

```
<?php

namespace MyApp\EventListener;

use Symfony\Component\Security\Http\Event\InteractiveLoginEvent;
use Symfony\Component\Security\Core\SecurityContext;
use Doctrine\Bundle\DoctrineBundle\Registry as Doctrine;

use Symfony\Component\DependencyInjection\ContainerAwareInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Custom login listener.
 */
class LoginListener implements ContainerAwareInterface
{
    /** @var \Symfony\Component\Security\Core\SecurityContext */
```

```
    private $securityContext;

    /** @var \Doctrine\ORM\EntityManager */
    private $em;

    /**
     * @param \Symfony\Component\Security\Core\SecurityContext
$securityContext
     * @param \Doctrine\Bundle\DoctrineBundle\Registry $doctrine
     */
    public function __construct(SecurityContext $securityContext,
Doctrine $doctrine)
    {
        $this->securityContext = $securityContext;
        $this->em = $doctrine->getEntityManager();
    }

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

    /**
     * @param
\Symfony\Component\Security\Http\Event\InteractiveLoginEvent $event
     */
    public function onSecurityInteractiveLogin(InteractiveLoginEvent
$event)
```

```
    {
        /**
         * @TODO: something if the user hasn't been on for a
significant period of
         *      time, so we can inform them of new features, etc etc.
         */
        $lastLoginInterval = sprintf('PT%sD',
$this->container->getParameter('activity_days'));

        if
($this->securityContext->isGranted('IS_AUTHENTICATED_FULLY')) {
            // user has just logged in
        }

        if
($this->securityContext->isGranted('IS_AUTHENTICATED_REMEMBERED')) {
            // user has logged in using remember_me cookie
        }

        /**
         * Update the lastOnlineDate every $this->activity_minutes
minutes.
         */
        $lastOnlineInterval = sprintf('PT%sM',
$this->container->getParameter('activity_minutes'));

        // do some other magic here
        $user = $event->getAuthenticationToken()->getUser();
        $now = new \DateTime();
        $lastOnlineDate = $user->getLastOnlineDate();
```

```
        $newActivityDate = (NULL === $lastOnlineDate) ? $now :
$lastOnlineDate->add(new \DateInterval($lastOnlineInterval));

        if ($now >= $newActivityDate) {
            $user->setLastOnlineDate($now);
            $this->em->persist($user);
            $this->em->flush();
        }

    }
}
```

app/config/services.yml

```
my_app.login_listener:
    class: 'MyApp\EventListener\LoginListener'
    arguments: ['@security.context', '@doctrine']
    tags:
        - { name: 'kernel.event_listener', event:
'security.interactive_login' }
    calls:
        - [ setContainer, [ @service_container ] ]
```

Alternatives

In this particular use-case, leveraging a setContainer call may not be necessary if all you're doing is setting a single value on a User entity. Alternatively, we could simply send the activity_minutes parameter as an argument to the service, and not use the container at all.

```
my_app.login_listener:
    class: 'MyApp\EventListener\LoginListener'
    arguments: ['@security.context', '@doctrine',
'%activity_minutes%', '%activity_days%']
    tags:
        - { name: 'kernel.event_listener', event:
'security.interactive_login' }
```

We'd have to adjust our LoginListener appropriately as well:

```
<?php

namespace MyApp\EventListener;

use Symfony\Component\Security\Http\Event\InteractiveLoginEvent;
use Symfony\Component\Security\Core\SecurityContext;
use Doctrine\Bundle\DoctrineBundle\Registry as Doctrine;

/**
 * Custom login listener.
 */
class LoginListener
{
    /** @var \Symfony\Component\Security\Core\SecurityContext */
    private $securityContext;

    /** @var \Doctrine\ORM\EntityManager */
    private $em;
    protected $activityMinutes;
    protected $activityDays;
```

```
    /**
     * @param \Symfony\Component\Security\Core\SecurityContext
$securityContext
     * @param \Doctrine\Bundle\DoctrineBundle\Registry $doctrine
     */
    public function __construct(SecurityContext $securityContext,
Doctrine $doctrine, $activityMinutes, $activityDays)
    {
        $this->securityContext = $securityContext;
        $this->em = $doctrine->getEntityManager();
        $this->activityMinutes = $activityMinutes;
        $this->activityDays = $activityDays;
    }

    /**
     * @param
\Symfony\Component\Security\Http\Event\InteractiveLoginEvent $event
     */
    public function onSecurityInteractiveLogin(InteractiveLoginEvent
$event)
    {
        /**
         * @TODO: something if the user hasn't been on for a
significant period of
         *      time, so we can inform them of new features, etc etc.
         */
        $lastLoginInterval = sprintf('PT%sD', $this->activityDays);

        if
($this->securityContext->isGranted('IS_AUTHENTICATED_FULLY')) {
            // user has just logged in
```

```
        }

        if
($this->securityContext->isGranted('IS_AUTHENTICATED_REMEMBERED')) {
            // user has logged in using remember_me cookie
        }

        /**
         * Update the lastOnlineDate every $this->activityMinutes
minutes.
         */
        $lastOnlineInterval = sprintf('PT%sM',
$this->activityMinutes);

        // do some other magic here
        $user = $event->getAuthenticationToken()->getUser();
        $now = new \DateTime();
        $lastOnlineDate = $user->getLastOnlineDate();

        $newActivityDate = (NULL === $lastOnlineDate) ? $now :
$lastOnlineDate->add(new \DateInterval($lastOnlineInterval));

        if ($now >= $newActivityDate) {
            $user->setLastOnlineDate($now);
            $this->em->persist($user);
            $this->em->flush();
        }

    }
}
```