

# How to manually set an authenticated user in Spring Security / SpringMVC

Asked 13 years, 11 months ago   Modified 4 months ago   Viewed 256k times



129



After a new user submits a 'New account' form, I want to manually log that user in so they don't have to login on the subsequent page.

The normal form login page going through the spring security interceptor works just fine.

In the new-account-form controller I am creating a UsernamePasswordAuthenticationToken and setting it in the SecurityContext manually:

```
SecurityContextHolder.getContext().setAuthentication(authentication);
```

On that same page I later check that the user is logged in with:

```
SecurityContextHolder.getContext().getAuthentication().getAuthorities();
```

This returns the authorities I set earlier in the authentication. All is well.

But when this same code is called on the very next page I load, the authentication token is just UserAnonymous.

I'm not clear why it did not keep the authentication I set on the previous request. Any thoughts?

- Could it have to do with session ID's not being set up correctly?
- Is there something that is possibly overwriting my authentication somehow?
- Perhaps I just need another step to save the authentication?
- Or is there something I need to do to declare the authentication across the whole session rather than a single request somehow?

Just looking for some thoughts that might help me see what's happening here.

java

authentication

spring-mvc

spring-security

Share Edit Follow

asked Jan 12, 2011 at 2:44



David Parks


31.9k ● 47 ● 199 ● 356

- 1 You can follow my answer to [stackoverflow.com/questions/4824395/...](https://stackoverflow.com/questions/4824395/...) – AlexK Nov 1, 2011 at 21:17
- 3 Readers, beware of the answers to this question if they tell you to do: `SecurityContextHolder.getContext().setAuthentication(authentication)`. It works, and is common, but there are serious functionality shortcomings that you will meet if you just do that. For more info, see my question, and the answer: [stackoverflow.com/questions/47233187/...](https://stackoverflow.com/questions/47233187/...) – goat Nov 28, 2017 at 2:47

Here is a related issue: [stackoverflow.com/questions/69681254/...](https://stackoverflow.com/questions/69681254/...) – Mircea Lutic Oct 22, 2021 at 18:06

Here is a related issue: [stackoverflow.com/questions/69681254/...](https://stackoverflow.com/questions/69681254/...) – Mircea Lutic Oct 22, 2021 at 18:07

## 9 Answers

Sorted by: Highest score (default) 



84



I couldn't find any other full solutions so I thought I would post mine. This may be a bit of a hack, but it resolved the issue to the above problem:

```
@Autowired
AuthenticationServiceImpl authenticationManager;

public void login(HttpServletRequest request, String userName, String
password) {

    UsernamePasswordAuthenticationToken authRequest = new
UsernamePasswordAuthenticationToken(userName, password);

    // Authenticate the user
    Authentication authentication =
authenticationManager.authenticate(authRequest);
    SecurityContext securityContext = SecurityContextHolder.getContext();
    securityContext.setAuthentication(authentication);

    // Create a new session and add the security context.
    HttpSession session = request.getSession(true);
    session.setAttribute("SPRING_SECURITY_CONTEXT", securityContext);
}
```

Share Edit Follow

edited Jan 30, 2023 at 14:52



James Jithin


10.5k ● 5 ● 39 ● 52

answered Dec 1, 2011 at 3:34



Stuart McIntyre

998 ● 6 ● 8

- 8 +1 - This helped me! I was missing the SPRING\_SECURITY\_CONTEXT update. ...But how "dirty" is this? – l3dx Nov 1, 2012 at 17:12 
- 23 where do you get `authenticationManager` from? – Isaac Nov 16, 2012 at 19:05
- 2 authenticationManager is autowired in your class like this `@Autowired AuthenticationServiceImpl authenticationManager`. And there must also be a bean injection in your xml configuration, so Spring knows what to inject. – user663381 Sep 12, 2013 at 10:30

- 4 Why is it necessary to create a new session? Doesn't SecurityContext handle that? – [Vlad Manuel Mureşan](#) Mar 16, 2016 at 14:37
- 3 I think people should be aware that doing this manual login via `setAuthentication()` bypasses some of the spring stuff, such as enforcing maximum concurrent sessions limit for a user, changing the users session id upon login to prevent session fixation, automatic registration of the session into a `SessionRegistry`, and maybe more. – [goat](#) Nov 10, 2017 at 23:42

68

I had the same problem as you a while back. I can't remember the details but the following code got things working for me. This code is used within a Spring Webflow flow, hence the RequestContext and ExternalContext classes. But the part that is most relevant to you is the doAutoLogin method.

```
public String registerUser(UserRegistrationFormBean userRegistrationFormBean,
    RequestContext requestContext,
    ExternalContext externalContext) {

    try {
        Locale userLocale = requestContext.getExternalContext().getLocale();
        this.userService.createNewUser(userRegistrationFormBean, userLocale,
            Constants.SYSTEM_USER_ID);
        String emailAddress =
            userRegistrationFormBean.getChooseEmailAddressFormBean().getEmailAddress();
        String password =
            userRegistrationFormBean.getChoosePasswordFormBean().getPassword();
        doAutoLogin(emailAddress, password, (HttpServletRequest)
            externalContext.getNativeRequest());
        return "success";
    } catch (EmailAddressNotUniqueException e) {
        MessageResolver messageResolvable
            = new MessageBuilder().error()

        .source(UserRegistrationFormBean.PROPERTYNAME_EMAIL_ADDRESS)

        .code("userRegistration.emailAddress.not.unique")
            .build();
        requestContext.getMessageContext().addMessage(messageResolvable);
        return "error";
    }
}

private void doAutoLogin(String username, String password, HttpServletRequest
    request) {

    try {
        // Must be called from request filtered by Spring Security, otherwise
        SecurityContextHolder is not updated
        UsernamePasswordAuthenticationToken token = new
            UsernamePasswordAuthenticationToken(username, password);
        token.setDetails(new WebAuthenticationDetails(request));
        Authentication authentication =
            this.authenticationProvider.authenticate(token);
        logger.debug("Logging in with [{}]", authentication.getPrincipal());
```

```

SecurityContextHolder.getContext().setAuthentication(authentication);
} catch (Exception e) {
    SecurityContextHolder.getContext().setAuthentication(null);
    logger.error("Failure in autoLogin", e);
}
}

```

Share Edit Follow

answered Jan 12, 2011 at 17:48



KevinS

7,879 ● 4 ● 41 ● 62

3 Thank you, the code is very helpful in helping me know that I'm troubleshooting in the right area. Looks like I have a smoking gun, it's creating a new session id after the manual authentication, but the old session id is still being identified from the cookie. Gotta figure out why now, but at least I'm clearly on track. Thanks! – [David Parks](#) Jan 13, 2011 at 0:47

4 Anyone following this guidance should also see this related issue:  
[stackoverflow.com/questions/4824395/...](http://stackoverflow.com/questions/4824395/...) – [David Parks](#) Jan 31, 2011 at 10:51

19 Can you please explain that how you are getting authenticationProvider – [vivex](#) May 8, 2015 at 5:28

1 @s1moner3d you should be able to get it injected via IoC -> \@Autowired – [Hartmut](#) Dec 12, 2016 at 11:07

2 @Configuration public class WebConfig extends WebSecurityConfigurerAdapter {  
 @Bean @Override public AuthenticationManager authenticationProvider() throws  
 Exception { return super.authenticationManagerBean(); } } – [slisnychyi](#) Aug 13,  
 2019 at 20:14 ✎



Ultimately figured out the root of the problem.

20



When I create the security context manually no session object is created. Only when the request finishes processing does the Spring Security mechanism realize that the session object is null (when it tries to store the security context to the session after the request has been processed).



At the end of the request Spring Security creates a new session object and session ID. However this new session ID never makes it to the browser because it occurs at the end of the request, after the response to the browser has been made. This causes the new session ID (and hence the Security context containing my manually logged on user) to be lost when the next request contains the previous session ID.

Share Edit Follow

answered Jan 13, 2011 at 1:57



David Parks

31.9k ● 47 ● 199 ● 356

6 Honestly this feels like a design flaw in spring security more than anything. There are plenty of frameworks written in other languages that would have no problem with this, but Spring Security just breaks.  
– [chubbsondubs](#) Dec 11, 2012 at 22:37

5 and the solution is? – [s1moner3d](#) Nov 18, 2016 at 14:26

3 and what is the solution? – [Thiago](#) May 5, 2017 at 19:59

One working solution in @KevinS answer above (see the code after the "Create a new session and add the security context." comment) – [Rolch2015](#) Jun 10, 2023 at 22:35



5



Turn on debug logging to get a better picture of what is going on.

You can tell if the session cookies are being set by using a browser-side debugger to look at the headers returned in HTTP responses. (There are other ways too.)

One possibility is that SpringSecurity is setting secure session cookies, and your next page requested has an "http" URL instead of an "https" URL. (The browser won't send a secure cookie for an "http" URL.)

Share Edit Follow

edited Dec 1, 2011 at 3:48

answered Jan 12, 2011 at 3:10



[Stephen C](#)

718k ● 95 ● 844 ● 1.3k

Thanks these were all very helpful and relevant suggestions! – [David Parks](#) Jan 13, 2011 at 0:48



5



The new filtering feature in Servlet 2.4 basically alleviates the restriction that filters can only operate in the request flow before and after the actual request processing by the application server. Instead, Servlet 2.4 filters can now interact with the request dispatcher at every dispatch point. This means that when a Web resource forwards a request to another resource (for instance, a servlet forwarding the request to a JSP page in the same application), a filter can be operating before the request is handled by the targeted resource. It also means that should a Web resource include the output or function from other Web resources (for instance, a JSP page including the output from multiple other JSP pages), Servlet 2.4 filters can work before and after each of the included resources. .

To turn on that feature you need:

### web.xml

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
```

```
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>*</strike></url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

## RegistrationController

```
return "forward:/login?j_username=" + registrationModel.getUserEmail()
      + "&j_password=" + registrationModel.getPassword();
```

Share Edit Follow

edited Feb 27, 2018 at 7:32

answered Nov 1, 2011 at 21:19



user8280225



AlexK

111 ● 2 ● 3

- 1 Good info, but putting the username and password into the url is bad. 1) no escaping is done, so a username or password with special character is likely to break, or even worse, be used as a security exploit vector. 2) passwords in urls are bad because urls often get logged to disk, which is pretty bad for security - all your passwords in plaintext just sitting there. – [goat](#) Jan 11, 2019 at 18:20

▲ I was trying to test an extjs application and after successfully setting a testingAuthenticationToken this suddenly stopped working with no obvious cause.

2 I couldn't get the above answers to work so my solution was to skip out this bit of spring in the test environment. I introduced a seam around spring like this:

```
public class SpringUserAccessor implements UserAccessor
{
    @Override
    public User getUser()
    {
        SecurityContext context = SecurityContextHolder.getContext();
        Authentication authentication = context.getAuthentication();
        return (User) authentication.getPrincipal();
    }
}
```

User is a custom type here.

I'm then wrapping it in a class which just has an option for the test code to switch spring out.

```
public class CurrentUserAccessor
{
    private static UserAccessor _accessor;

    public CurrentUserAccessor()
    {
        _accessor = new SpringUserAccessor();
    }
}
```

```

    }

    public User getUser()
    {
        return _accessor.getUser();
    }

    public static void UseTestingAccessor(User user)
    {
        _accessor = new TestUserAccessor(user);
    }
}

```

The test version just looks like this:

```

public class TestUserAccessor implements UserAccessor
{
    private static User _user;

    public TestUserAccessor(User user)
    {
        _user = user;
    }

    @Override
    public User getUser()
    {
        return _user;
    }
}

```

In the calling code I'm still using a proper user loaded from the database:

```

User user = (User) _userService.loadUserByUsername(username);
CurrentUserAccessor.UseTestingAccessor(user);

```

Obviously this won't be suitable if you actually need to use the security but I'm running with a no-security setup for the testing deployment. I thought someone else might run into a similar situation. This is a pattern I've used for mocking out static dependencies before. The other alternative is you can maintain the staticness of the wrapper class but I prefer this one as the dependencies of the code are more explicit since you have to pass `CurrentUserAccessor` into classes where it is required.

Share Edit Follow

edited Jun 19, 2014 at 11:26

answered Jun 19, 2014 at 11:17



JonnyRaa

7,978 ● 6 ● 48 ● 50

1 why not just disable security in test env, if that's what you intend to do – eis Jun 3, 2021 at 11:25



## Spring Security 6 + update

0

The answer Stuart provider works with Spring Security 6 but it's potentially unreliable to manually link the security context to the HTTP session by manually setting it to the attribute `SPRING_SECURITY_CONTEXT`, which can change in the future.



The way I would, [base on the official document from Spring Security](#), is to use the `SecurityContextHolder`. Note that the `SecurityContextHolder` stays private to the login controller code. It's not a singleton and does not need to be injected anywhere else.

```
private SecurityContextHolder securityContextHolder =
    new HttpSessionSecurityContextHolder();

@PostMapping("/login")
public void login(HttpServletRequest request, HttpServletResponse response) {

    // Authenticate the login request here and build the Authenticate object
    Authentication authentication = new UsernamePasswordAuthenticationToken(user,
    accessToken, authorities);

    // Create empty security context and set authentication
    SecurityContext context = securityContextHolder.createEmptyContext();
    context.setAuthentication(authentication);
    securityContextHolder.setContext(context);

    // Save the security context to the repo (This adds it to the HTTP session)
    securityContextHolder.saveContext(context, request, response);
}
```

Prior to the new Spring Security, the code above can be achieved by a single line of code below.

After upgrading to the new security, you will see debug logs from

`AnonymousAuthenticationFilter` saying `Set SecurityContextHolder to anonymous` `SecurityContext` constantly, causing the future HTTP requests to have anonymous context (and of course, causing auth to fail or restart the auth process again).

```
SecurityContextHolder.getContext().setAuthentication(authentication);
```

Share Edit Follow

edited Jul 10 at 16:10

answered Jul 10 at 15:48



Fangming

25.2k ● 7 ● 104 ● 92



0

It's been a while since the answers were updated. Spring Boot 3, and Spring Security 6 has come out. While I found that the accepted answer still works, the Spring documentation contains notes on how to manually store and remove authentication in the Spring Security Context.



[Spring Security Docs - Storing Authentication Manually](#)





Basically,



```
private SecurityContextRepository securityContextRepository =
    new HttpSessionSecurityContextRepository();

@PostMapping("/login")
public void login(@RequestBody LoginRequest loginRequest, HttpServletRequest
request, HttpServletResponse response) {
    UsernamePasswordAuthenticationToken token =
    UsernamePasswordAuthenticationToken.unauthenticated(
        loginRequest.getUsername(), loginRequest.getPassword());
    Authentication authentication = authenticationManager.authenticate(token);
    SecurityContext context = securityContextHolderStrategy.createEmptyContext();
    context.setAuthentication(authentication);
    securityContextHolderStrategy.setContext(context);
    securityContextRepository.saveContext(context, request, response);
}
```

Share Edit Follow

answered Mar 28 at 20:46

klala  
39 ● 5

0



I'm as surprised as you are. This keeps happening for some reason. I don't think it should be made normal. Springboot should do this and not us. Since if there is any other configuration behind the scenes, we will not know how to manage it correctly as well as spring would. Therefore, we can say that it is a bug and must be done manually.

```
@RequestMapping(value = {"/login"}, method = RequestMethod.POST)
public ResponseEntity << ? > getUserLogin(@RequestBody UserLoginDTO
userLoginDTO, HttpServletRequest request) {

    try {
        // Calls your SecurityConfig AuthenticationManager (...) ->
        authenticationProvider()
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(userLoginDTO.getEmail(),
            userLoginDTO.getPassword()));

        HttpSession session = request.getSession(true);
        SecurityContextHolder.getContext().setAuthentication(authentication);

        session.setAttribute(HttpSessionSecurityContextRepository.SPRING_SECURITY_CONTEXT_KEY,
            SecurityContextHolder.getContext());

        org.springframework.security.core.userdetails.User authenticatedUser =
            (org.springframework.security.core.userdetails.User) authentication
                .getPrincipal();

        if (authenticatedUser != null) {
            return your response
        } {
            return your response
        } catch (UserNotFoundException e) {
            return your response
        } catch (Exception e) {
```

```
        return your response  
    }  
}
```

Will you have problems with the csrf? I have no idea.

Will you have problems when it comes to sessions? I have no idea.

...

[Share](#) [Edit](#) [Follow](#)

[edited Jul 17 at 6:36](#)

[answered Jul 17 at 6:30](#)



[Drakgoku](#)

1 ● 5