YADeB

Yet Another DEveloper's Blog

Make Spring Security Context Available Inside A Hystrix Command

☐ September 15, 2014 ☐ jfconavarrete ☐ java ☐ Tags: AuthenticationCredentialsNotFoundException, hystrix, hystrixrequestcontext, java, spring, springsecurity

Problem

When a service secured by SpringSecurity is executed inside a Hystrix Command you will get the following exception:

```
lsNotFoundException : An Authentication object was not found in the SecurityContext
```

This is because Hystrix is running the command in a thread from its own thread pool (which we'll call thread H)and not in the thread which follows the normal course of execution (which we'll call thread A). The default strategy used by the SecurityContextHolder is ThreadLocalSecurityContextHolderStrategy; which, as it's name implies, is based on ThreadLocal. This means that the SecurityContext will be available on only on thread A and not in thread H. So, when we try to access the SecurityContext via SecurityContextHolder in thread H, we will get the AuthenticationCredentialsNotFoundException.

Solution

The solution presented here is for a web application.

Create the HystrixRequestContext

The HystrixRequestContext must be initialized at the beginning of each request before RequestVariable functionality can be used. This enables the HystrixRequestContext to contain the state and manage the lifecycle of HystrixRequestVariableDefault objects that provide request scoped (rather than only thread

scoped) variables so that multiple threads within a single request can share state.

```
/**
 * ServletFilter for initializing HystrixRequestContext at the beginning of an
 *
 * The filter shuts down the HystrixRequestContext at the end of the request to
 * leakage into subsequent requests.
 */
public class HystrixRequestContextEnablerFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, Filt
        HystrixRequestContext context = HystrixRequestContext.initializeContext
        try {
            chain.doFilter(request, response);
        } finally {
            context.shutdown();
        }
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void destroy() {}

}
```

Create a HystrixRequestVariable for the SecurityContext

The SecurityContextHystrixRequestVariable is a holder which will allow us to access the Spring Security SecurityContext. The HystrixRequestVariableDefault implementation is able to get the right SecurityContext for concurrent requests as long as the same instance of HystrixRequestVariableDefault is used.

```
/**
 * Holder for the HystrixRequestVariableDefault instance that contains the Secu
 */
public class SecurityContextHystrixRequestVariable {

    private static final HystrixRequestVariableDefault<SecurityContext> securit

    private SecurityContextHystrixRequestVariable() {}

    public static HystrixRequestVariableDefault<SecurityContext> getInstance()
        return securityContextVariable;
    }

}
```

Set the SecurityContext in the HystrixRequestVariable

Now we need to create a Filter which makes the Spring SecurityContext available in Hystrix's threads by including it in thHystrixRequestVariable by using the class we created in the previous step. This filter must run after the HystrixRequestContext has been initialized.

```
public class SecurityContextHystrixRequestVariableSetterFilter implements Filt

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, Filt
        SecurityContextHystrixRequestVariable.getInstance().set(SecurityContext

        chain.doFilter(request, response);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void destroy() {}

}
```

Set the SecurityContext in the SecurityContextHolder

Create a command hook that extracts the SecurityContext from the SecurityContextHystrixRequestVariable and sets it on the SecurityContextHolder. The hook will set the SecurityContext when the Runnable is about to start and when the Runnable has completed successfully or unsucessfuly.

```
/**
 * A HystrixCommandExecutionHook that makes the Spring SecurityContext availabl
 *
 * It extracts the SecurityContext from the SecurityContextHystrixRequestVariab
 */
public class SecurityContextRegistratorCommandHook extends HystrixCommandExecut

    @Override
    public <T> void onRunStart(HystrixCommand<T> commandInstance) {
        SecurityContextHolder.setContext(SecurityContextHystrixRequestVariable
    }

    /**
     * Clean the SecurityContext
     */
    @Override
    public <T> T onComplete(HystrixCommand<T> commandInstance, T response) {
        SecurityContextHolder.setContext(null);

        return response;
    }

    /**
     * Clean the SecurityContext
     */
    @Override
    public <T> Exception onError(HystrixCommand<T> commandInstance, FailureType
        SecurityContextHolder.setContext(null);

        return e;
    }

}
```

Configure the Filters and the Command Hook

To make Hystrix use our command hook we have to run the following statement on start up:

```
1 HystrixPlugins.getInstance().registerCommandExecutionHook(new SecurityContextRe
```

We must register our filters after the SpringSecurityFilterChain so that the SecurityContext is already created. We can do that easily with Spring's Java Config.

```
1  @Configuration
2  public static class SecurityWebApplicationInitializer extends AbstractSecurityI
3
4      /**
5      * Add the HystrixRequestContextEnablerFilter and the SecurityContextHystri
6      */
7      @Override
8      protected void afterSpringSecurityFilterChain(ServletContext servletContex
9          servletContext.addFilter("hystrixRequestContextEnablerFilter", new Hys
10         .addMappingForUrlPatterns(null, true, "/*");
11
12         servletContext.addFilter("hystrixSecurityContextEnablerFilter", new Se
13             .addMappingForUrlPatterns(null, true, "/*");
14     }
15
16 }
```

The End

Now, the exception should be gone and you should be able to access the SecurityContext inside a HystrixCommand.

One thought on “Make Spring Security Context Available Inside A Hystrix Command”

Figured out finally. In Spring Boot Resource Server, extends AbstractSecurityWebApplicationInitializer won’t work, guess springboot just pick the default one with default resource server configuration.
And using FilterRegistrationBean to adjust the Filter Order works.

```
@Bean
public FilterRegistrationBean
securityFilterChain(@Qualifier(AbstractSecurityWebApplicationInitializer.DEFAULT_FILTER_NAME)
Filter securityFilter) {
FilterRegistrationBean registration = new FilterRegistrationBean(securityFilter);
registration.setOrder(Integer.MAX_VALUE – 2);
registration.setName(AbstractSecurityWebApplicationInitializer.DEFAULT_FILTER_NAME);
return registration;
}

@Bean
public FilterRegistrationBean userInsertingMdcFilterRegistrationBean() {
FilterRegistrationBean registrationBean = new FilterRegistrationBean();
HystrixRequestContextEnablerFilter userFilter = new HystrixRequestContextEnablerFilter();
registrationBean.setFilter(userFilter);
```

```
registrationBean.setOrder(Integer.MAX_VALUE – 1);
return registrationBean;
}

@Bean
public FilterRegistrationBean setterInsertingMdcFilterRegistrationBean() {
FilterRegistrationBean registrationBean = new FilterRegistrationBean();
SecurityContextHystrixRequestVariableSetterFilter userFilter = new
SecurityContextHystrixRequestVariableSetterFilter();
registrationBean.setFilter(userFilter);
registrationBean.setOrder(Integer.MAX_VALUE);
return registrationBean;
}
```

Reply

Frank
August 9, 2016 at 12:45 pm