Shiro Security

Shiro Security Component

Available as of Camel 2.5

The **shiro-security** component in Camel is a security focused component, based on the Apache Shiro security project.

Apache Shiro is a powerful and flexible open-source security framework that cleanly handles authentication, authorization, enterprise session management and cryptography. The objective of the Apache Shiro project is to provide the most robust and comprehensive application security framework available while also being very easy to understand and extremely simple to use.

This camel shiro-security component allows authentication and authorization support to be applied to different segments of a camel route.

Shiro security is applied on a route using a Camel Policy. A Policy in Camel utilizes a strategy pattern for applying interceptors on Camel Processors. It offering the ability to apply cross-cutting concerns (for example. security, transactions etc) on sections/segments of a camel route.

Maven users will need to add the following dependency to their `pom.xml` for this component:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-shiro</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

Shiro Security Basics

To employ Shiro security on a camel route, a ShiroSecurityPolicy object must be instantiated with security configuration details (including users, passwords, roles etc). This object must then be applied to a camel route. This ShiroSecurityPolicy Object may also be registered in the Camel registry (JNDI or ApplicationContextRegistry) and then utilized on other routes in the Camel Context.

Configuration details are provided to the ShiroSecurityPolicy using an Ini file (properties file) or an Ini object. The Ini file is a standard Shiro configuration file containing user/role details as shown below

```
[users]
# user 'ringo' with password 'starr' and the 'sec-level1' role
ringo = starr, sec-level1
george = harrison, sec-level2
john = lennon, sec-level3
paul = mccartney, sec-level3

[roles]
# 'sec-level3' role has all permissions, indicated by the
# wildcard '*'
sec-level3 = *

# The 'sec-level2' role can do anything with access of permission
# readonly (*) to help
sec-level2 = zone1:*

# The 'sec-level1' role can do anything with access of permission
# readonly
sec-level1 = zone1:readonly:*
```

Instantiating a ShiroSecurityPolicy Object

A ShiroSecurityPolicy object is instantiated as follows

```
        private final String iniResourcePath = "classpath:shiro.ini";
        private final byte[] passPhrase = {
            (byte) 0x08, (byte) 0x09, (byte) 0x0A, (byte) 0x0B,
            (byte) 0x0C, (byte) 0x0D, (byte) 0x0E, (byte) 0x0F,
            (byte) 0x10, (byte) 0x11, (byte) 0x12, (byte) 0x13,
            (byte) 0x14, (byte) 0x15, (byte) 0x16, (byte) 0x17};
        List<permission> permissionsList = new ArrayList<permission>();
        Permission permission = new WildcardPermission("zone1:readwrite:*");
        permissionsList.add(permission);

        final ShiroSecurityPolicy securityPolicy =
            new ShiroSecurityPolicy(iniResourcePath, passPhrase, true, permissionsList);
```

ShiroSecurityPolicy Options

Name	Default Value	Type	Description
`iniResourcePath or ini`	`none`	Resource String or Ini Object	A mandatory Resource String for the iniResourcePath or an instance of an Ini object must be passed to the security policy. Resources can be acquired from the file system, classpath, or URLs when prefixed with "file:, classpath:, or url:" respectively. For e.g "classpath:shiro.ini"
`passPhrase`	`An AES 128 based key`	byte[]	A passPhrase to decrypt ShiroSecurityToken(s) sent along with Message Exchanges
`alwaysReauthenticate`	`true`	boolean	Setting to ensure re-authentication on every individual request. If set to false, the user is authenticated and locked such than only requests from the same user going forward are authenticated.
`permissionsList`	`none`	List<Permission>	A List of permissions required in order for an authenticated user to be authorized to perform further action i.e continue further on the route. If no Permissions list or Roles List (see below) is provided to the ShiroSecurityPolicy object, then authorization is deemed as not required. Note that the default is that authorization is granted if any of the Permission Objects in the list are applicable.
`rolesList`	`none`	List<String>	**Camel 2.13:** A List of roles required in order for an authenticated user to be authorized to perform further action i.e continue further on the route. If no roles list or permissions list (see above) is provided to the ShiroSecurityPolicy object, then authorization is deemed as not required. Note that the default is that authorization is granted if any of the roles in the list are applicable.
`cipherService`	`AES`	org.apache.shiro.crypto.CipherService	Shiro ships with AES & Blowfish based CipherServices. You may use one these or pass in your own Cipher implementation
`base64`	`false`	`boolean`	**Camel 2.12:** To use base64 encoding for the security token header, which allows transferring the header over JMS etc. This option must also be set on `ShiroSecurityTokenInjector` as well.
`allPermissionsRequired`	`false`	`boolean`	**Camel 2.13:** The default is that authorization is granted if any of the Permission Objects in the permissionsList parameter are applicable. Set this to true to require all of the Permissions to be met.
`allRolesRequired`	`false`	`boolean`	**Camel 2.13:** The default is that authorization is granted if any of the roles in the rolesList parameter are applicable. Set this to true to require all of the roles to be met.

Applying Shiro Authentication on a Camel Route

The ShiroSecurityPolicy, tests and permits incoming message exchanges containing a encrypted SecurityToken in the Message Header to proceed further following proper authentication. The SecurityToken object contains a Username/Password details that are used to determine where the user is a valid user.

```
protected RouteBuilder createRouteBuilder() throws Exception {
    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy("classpath:shiro.ini", passPhrase);

    return new RouteBuilder() {
        public void configure() {
            onException(UnknownAccountException.class).
                to("mock:authenticationException");
            onException(IncorrectCredentialsException.class).
                to("mock:authenticationException");
            onException(LockedAccountException.class).
                to("mock:authenticationException");
            onException(AuthenticationException.class).
                to("mock:authenticationException");

            from("direct:secureEndpoint").
                to("log:incoming payload").
                policy(securityPolicy).
                to("mock:success");
        }
    };
}
```

Applying Shiro Authorization on a Camel Route

Authorization can be applied on a camel route by associating a Permissions List with the ShiroSecurityPolicy. The Permissions List specifies the permissions necessary for the user to proceed with the execution of the route segment. If the user does not have the proper permission set, the request is not authorized to continue any further.

```
protected RouteBuilder createRouteBuilder() throws Exception {
    final ShiroSecurityPolicy securityPolicy =
        new ShiroSecurityPolicy("./src/test/resources/securityconfig.ini", passPhrase);

    return new RouteBuilder() {
        public void configure() {
            onException(UnknownAccountException.class).
                to("mock:authenticationException");
            onException(IncorrectCredentialsException.class).
                to("mock:authenticationException");
            onException(LockedAccountException.class).
                to("mock:authenticationException");
            onException(AuthenticationException.class).
                to("mock:authenticationException");

            from("direct:secureEndpoint").
                to("log:incoming payload").
                policy(securityPolicy).
                to("mock:success");
        }
    };
}
```

Creating a ShiroSecurityToken and injecting it into a Message Exchange

A ShiroSecurityToken object may be created and injected into a Message Exchange using a Shiro Processor called ShiroSecurityTokenInjector. An example of injecting a ShiroSecurityToken using a ShiroSecurityTokenInjector in the client is shown below

```
ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "starr");
ShiroSecurityTokenInjector shiroSecurityTokenInjector =
    new ShiroSecurityTokenInjector(shiroSecurityToken, passPhrase);

from("direct:client").
    process(shiroSecurityTokenInjector).
    to("direct:secureEndpoint");
```

Sending Messages to routes secured by a ShiroSecurityPolicy

Messages and Message Exchanges sent along the camel route where the security policy is applied need to be accompanied by a SecurityToken in the Exchange Header. The SecurityToken is an encrypted object that holds a Username and Password. The SecurityToken is encrypted using AES 128 bit security by default and can be changed to any cipher of your choice.

Given below is an example of how a request may be sent using a ProducerTemplate in Camel along with a SecurityToken

```
    @Test
    public void testSuccessfulShiroAuthenticationWithNoAuthorization() throws Exception {
        //Incorrect password
        ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "stirr");

        // TestShiroSecurityTokenInjector extends ShiroSecurityTokenInjector
        TestShiroSecurityTokenInjector shiroSecurityTokenInjector =
            new TestShiroSecurityTokenInjector(shiroSecurityToken, passPhrase);

        successEndpoint.expectedMessageCount(1);
        failureEndpoint.expectedMessageCount(0);

        template.send("direct:secureEndpoint", shiroSecurityTokenInjector);

        successEndpoint.assertIsSatisfied();
        failureEndpoint.assertIsSatisfied();
    }
```

Sending Messages to routes secured by a ShiroSecurityPolicy (much easier from Camel 2.12 onwards)

From **Camel 2.12** onwards its even easier as you can provide the subject in two different ways.

Using ShiroSecurityToken

You can send a message to a Camel route with a header of key `ShiroSecurityConstants.SHIRO_SECURITY_TOKEN` of the type `org.apache.camel.component.shiro.security.ShiroSecurityToken` that contains the username and password. For example:

```
        ShiroSecurityToken shiroSecurityToken = new ShiroSecurityToken("ringo", "starr");

        template.sendBodyAndHeader("direct:secureEndpoint", "Beatle Mania", ShiroSecurityConstants.
SHIRO_SECURITY_TOKEN, shiroSecurityToken);
```

You can also provide the username and password in two different headers as shown below:

```
        Map<String, Object> headers = new HashMap<String, Object>();
        headers.put(ShiroSecurityConstants.SHIRO_SECURITY_USERNAME, "ringo");
        headers.put(ShiroSecurityConstants.SHIRO_SECURITY_PASSWORD, "starr");
        template.sendBodyAndHeaders("direct:secureEndpoint", "Beatle Mania", headers);
```

When you use the username and password headers, then the ShiroSecurityPolicy in the Camel route will automatic transform those into a single header with key ShiroSecurityConstants.SHIRO_SECURITY_TOKEN with the token. Then token is either a `ShiroSecurityToken` instance, or a base64 representation as a String (the latter is when you have set base64=true).