

# Ambari via KnoxSSO and Default IDP

## Form-based Authentication for Existing Hadoop Web Applications

### Introduction

Apache Knox with default KnoxSSO IDP enables the use of form based authentication as a SSO solution for accessing and developing KnoxSSO enabled applications including, Ambari, Ranger, Hadoop UIs and custom built applications that utilize REST APIs through Knox. These capabilities will be available in the Knox 0.9.0 release.

This paper illustrates the use of the default IDP for form based authentication in Apache Knox. The same sort of flow that is described below would be available for Ranger, Hadoop UIs or any KnoxSSO participating application.

### Form-based Authentication (Default IDP)

Available in Apache Knox 0.9.0 is the new ability to host web applications. This capability will be leveraged to deliver applications out of the box with Apache Knox and can be leveraged to deploy custom by customers and integrators. The first application to be delivered along with Apache Knox is a default IDP for KnoxSSO that provides form-based authentication for participating applications. This allows applications that are in KnoxSSO to easily integrate a form-based authentication using the application called *knoxauth* and LDAP or Active Directory.



Form-based Authentication with the KnoxSSO application - *knoxauth*.

The *knoxauth* application depends on the ShiroProvider and its LDAP/AD integration for authenticating HTTP Basic Auth credentials. It also requires certain configuration settings on the ShiroProvider that are not normally used for Shiro. We will cover those in detail in the sections that follow.


As always with KnoxSSO, participating applications need only ever configure the URL to the KnoxSSO/webssso endpoint - ie. <https://gatewayhost:8443/gateway/knoxssso/api/v1/webssso>. The ShiroProvider needs to be configured to redirect an incoming request with no Authorization header to the URL configured for the *redirectToUrl* provider param. This will be the relative url to the *knoxauth* application within the *knoxssso* topology - */gateway/knoxssso/knoxauth/login.html*. Again, this URL is hidden from the configuration of the participating applications in order to simplify changes from one IDP to another without having to change the world. The out-of-box *knoxssso.xml* topology is already configured for this redirect and other nuances required for the KnoxSSO Default IDP.

### Tutorial

#### Prerequisites

1. Follow the Ambari Vagrant Quick Start guide (<https://cwiki.apache.org/confluence/display/AMBARI/Quick+Start+Guide>) to create a three node cluster with Centos 6.4 using Ambari 2.4 (trunk) or greater
2. Unzip the Apache Knox v0.9.0 release candidate to the {AMBARI\_VAGRANT\_HOME}/centos6.4 directory which is a shared volume inside the vagrant machine /vagrant
3. vagrant ssh into c6401
4. Stop the Apache Knox instance that is already running (if there is one)
5. The *knoxssso.xml* topology file should be deployed by default and can be found at {GATEWAY\_HOME}/conf/topologies/*knoxssso.xml* file. The LDAP/AD configuration will need to be changed to match your deployment scenario. By default it is configured for the Knox Demo LDAP server - just as *sandbox.xml* is.
6. Change the *knoxssso.cookie.secure.only* param in *knoxssso.xml* to false. Ambari does not have SSL enabled by default and if we set the cookie to secure only it will not be presented to Ambari by the browser. **NOTE: THIS IS INSECURE AND ONLY USED FOR TESTING**

7. Change the `knoxsso.token.ttl` parameter to something like 30 secs (30000) - the default value of -1 will not work with Ambari until

 **AMBAR-15479** - JwtAuthenticationFilter needs to accommodate null JWT expiration time **RESOLVED** is resolved.

8. Ensure that the `knoxsso.redirect.whitelist.regex` parameter for KNOXSSO includes the `c6401.ambari.apache.org` host in the set of acceptable hosts. See the example at the end of this article.
9. Start your v0.9.0 version of Knox via: `{GATEWAY_HOME}/bin/gateway.sh start` - ensure that you are using Java 7 or 8 by exporting `JAVA_HOME` appropriately
10. Configure Ambari for SSO with KnoxSSO through the SSO Wizard via the `ambari-server CLI`
  - a. Get the gateway-identity public key from Apache Knox `{GATEWAY_HOME}/data/security/keystores/gateway.jks` via `keytool` or `portecle` (see Extracting Knox Public Key for IdP Configuration section for details)
  - b. Get the SSO provider URL for the KnoxSSO websso endpoint (i.e. <https://c6401.ambari.apache.org:8443/gateway/knoxsso/api/v1/websso>)
  - c. `su` to root `{pw: vagrant}`
  - d. start the sso wizard:

```
[root@c6401 Knox-0.9.0]# ambari-server setup-sso
```

```
Using python /usr/bin/python2
```

```
Setting up SSO authentication properties...
```

```
Do you want to configure SSO authentication [y/n] (y)?y
```

```
Provider URL [URL]: https://c6401.ambari.apache.org:8443/gateway/knoxsso/api/v1/websso
```

```
Public Certificate pem (stored) (empty line to finish input):
```

```
MIICOjCCAaOgAwIBAgIJANjgCshp4cP2MA0GCSqGSIb3DQEBBQUAMF8xCzAJBgNV
BAYTALVTMQ0wCwYDVQQIEWRUZXR0Mj0wCwYDVQQHEWRUZXR0Mj0wCwYDVQQKEWZl
YWRvb3AxDTALBgNVBAsTBFRlc3QxejAQBGNVBAMTCWxvY2FsaG9zdDAAeFw0xNjAx
MzAxNDUwMTNaFw0xNzAxMj0xNDUwMTNaMF8xCzAJBgNVBAYTALVTMQ0wCwYDVQQIE
WRUZXR0Mj0wCwYDVQQHEWRUZXR0Mj0wCwYDVQQKEWZlYWRvb3AxDTALBgNVBAsT
BFRlc3QxejAQBGNVBAMTCWxvY2FsaG9zdDABnzANBgkqhkiG9w0BAQEFAAOBjQAw
gYkCgYEAicKDXg/OXk1B1tlylj0PMvNpZc4cMagX6P20EryzLQpMMagYVbpbL0zU
D3B3M86gEFIFmgebJ95v8EydB5g4CIbF3CmDORK77Fy265xLXv06bhVqjvU1Q+zg
gwu8YeH9ZQcgfCaDKG3Corb8mu3W9JBNYkdqrAxMsIeLvu4ASVUCAwEAATANBgkq
hkiG9w0BAQUFAAOBgQA0xEgGw1Ho6DXMYVJuyxQYyN/0/NWKZ2Ysrx7MwZd3pJBT
XVN7K9jnr4qFrw4ok2yHWQszijUPhPxZqGZcKf6+ATjoUIXV+UiX+Nj0R2ov/JQ
fDfhTSQIXakBKK3z/3q5/iQSFdsqIgI/Ce7im9GsFesMhJnyk9aXb8ZROx4hvQ==
```

```
Do you want to configure advanced properties [y/n] (n) ?n
```

```
Ambari Server 'setup-sso' completed successfully.
```

1. Restart Ambari server:

```
[root@c6401 Knox-0.9.0]# ambari-server restart
```

## Extracting Knox Public Key for IdP Configuration

There are multiple ways that you can do this.

The following will use `keytool` to extract a PEM encoded cert from the gateway keystore:

```
[vagrant@c6401 Knox-0.9.0]$ keytool -exportcert -keystore data/security/keystores/gateway.jks -alias gateway-identity -rfc -file gateway.pem
```

```
Enter keystore password:{master secret}
```

```
Certificate stored in file <gateway.pem>
```

For the Ambari SSO wizard the content between

—BEGIN CERTIFICATE— and —END CERTIFICATE—

must be provided when requested. This is by the Ambari KnoxSSO integration point for verification of the SSO tokens issued by KnoxSSO.

## Portecle

The free Portecle tool is great for extracting PEM encoded certs when the process doesn't need to be automated. It may also be used when the keytool doesn't allow passwords that are shorter than 6 chars to be used. Even when the keystore actually exists with shorter passwords <sigh>.

[blocked URL](#)

## Apache Ambari

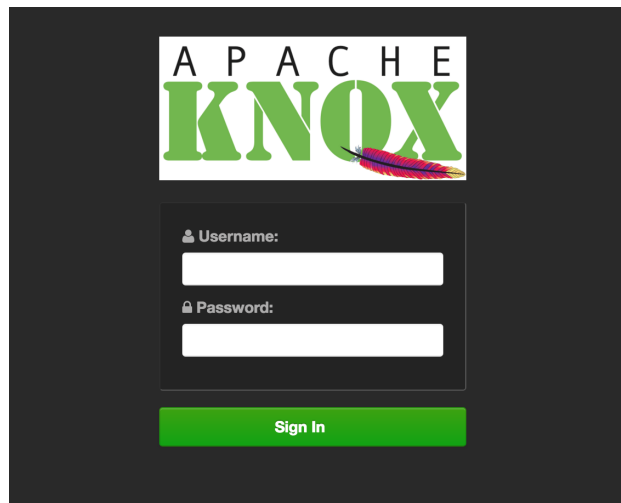
To demonstrate the integration between KnoxSSO and the new Default IDP (form-based IDP) for existing KnoxSSO aware Hadoop applications, Ambari will be used. This demonstrates Ambari's ability to acquire and validate KnoxSSO tokens/cookies as a means to authenticate to its management capabilities and custom views.

Once logged in through KnoxSSO the resulting hadoop-jwt cookie is used to create an Ambari session. Apache Ambari only knows that it is relying on KnoxSSO and nothing about the underlying SSO provider (in this case LDAP/AD and the Default IDP).

## Test Integration with Default IDP

1. Open Apache Ambari in a browser at <http://c6401.ambari.apache.org:8080> - you will initially be presented the Ambari login page but quickly redirected to the KnoxSSO login.

[blocked URL](#)



2. When presented with a login form, fill it out with these credentials (guest/guest-password) and submit it to the KnoxSSO. This will result in a HTTP Basic Auth credentials POST to the KnoxSSO/webssso endpoint. The credentials will be verified against the configured LDAP/AD identity store via the shiro provider and the authenticated identity normalized into a Java Subject. The successful authentication continues the processing through the provider chain and the identity assertion provider may use appropriate principal mapping to establish the effective username. The effective username is what the KnoxSSO service will put into the JWT token to be presented as a Cookie to all participating applications.

4. After a brief "Loading..." page you should be redirected back to Ambari. If you are interested you may find the hadoop-jwt cookie using Chrome's developer tools. It should be a session cookie set as HttpOnly and (normally) Secure. The service parameter `knoxsso.cookie.secure.only` for the KnoxSSO service in the `knoxsso.xml` topology controls the secure only setting of the cookie.

[blocked URL](#)

Note how Ambari accepts successful authentication even when they are not existing users. The user is added to the Ambari database and they are assigned minimum privileges. As you can see above the authentication of guest was successful and they have been granted rights to their custom views - of which there are none.

An existing user with normal privileges would now have access to all of the Ambari capabilities and views for which they are permitted.

## Topologies

The contents of these topology files can be copied into your {GATEWAY\_HOME}/conf/topologies directory.

### knoxsso.xml

The knoxsso.xml topology describes the manner in which a client acquires a KnoxSSO websso cookie/token. The shiro provider allows the integration LDAP/AD with HTTP Basic Auth credentials.

```
<topology>
  <gateway>
    <provider>
      <role>webappsec</role>
      <name>WebAppSec</name>
      <enabled>true</enabled>
      <param>
        <name>xframe.options.enabled</name>
        <value>true</value>
      </param>
    </provider>
    <provider>
      <role>authentication</role>
      <name>ShiroProvider</name>
      <enabled>true</enabled>
      <param>
        <name>sessionTimeout</name>
        <value>30</value>
      </param>
      <param>
        <name>redirectToUrl</name>
        <value>/gateway/knoxsso/knoxauth/login.html</value>
      </param>
      <param>
        <name>restrictedCookies</name>
        <value>rememberme,WWW-Authenticate</value>
      </param>
      <param>
        <name>main.LdapRealm</name>
```

```

    <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</value>
  </param>
  <param>
    <name>main.LdapContextFactory</name>
    <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
  </param>
  <param>
    <name>main.LdapRealm.contextFactory</name>
    <value>$LdapContextFactory</value>
  </param>
  <param>
    <name>main.LdapRealm.userDnTemplate</name>
    <value>uid={0},ou=people,dc=hadoop,dc=apache,dc=org</value>
  </param>
  <param>
    <name>main.LdapRealm.contextFactory.url</name>
    <value>ldap://localhost:33389</value>
  </param>
  <param>
    <name>main.LdapRealm.authenticationCachingEnabled</name>
    <value>false</value>
  </param>
  <param>
    <name>main.LdapRealm.contextFactory.authenticationMechanism</name>
    <value>simple</value>
  </param>
  <param>
    <name>urls./**</name>
    <value>authcBasic</value>
  </param>
</provider>
<provider>
  <role>identity-assertion</role>
  <name>Default</name>
  <enabled>true</enabled>
</provider>
<provider>
  <role>hostmap</role>
  <name>static</name>
  <enabled>true</enabled>

```

```
    <param><name>localhost</name><value>sandbox,<a href="http://sandbox.hortonworks.com">sandbox.hortonworks.com</a></value></param>

  </provider>

</gateway>

<application>

  <name>knoauth</name>

</application>

<service>

  <role>KNOXSSO</role>

  <param>

    <name>knoxsso.cookie.secure.only</name>

    <value>>false</value>

  </param>

  <param>

    <name>knoxsso.token.ttl</name>

    <value>30000</value>

  </param>

  <param>

    <name>knoxsso.redirect.whitelist.regex</name>

    <value>^https?:VV(c64\dl\ambari\.apache\.org|localhost|127\.\0\.\0\.\1|0:0:0:0:0:0:0:1|::1):[0-9].*$</value>

  </param>

</service>

</topology>
```