Readme File for IBM® Spectrum Symphony RFE 97838

Readme file for: IBM® Spectrum Symphony
Product/Component Release: 7.1.2
Fix ID: sym-7.1.2-build450098-jpmc
Publication date: Nov 22, 2017

This enhancement provides three authentication APIs that enable you to implement your own authentication logic and build a custom plug-in for authentication in a cluster where both IBM Spectrum Symphony 7.1.2 and IBM Spectrum Conductor with Spark 2.2.0 are installed.

Scope

Before you install this update to your cluster, note the following requirements:

Applicability	
Operating system	RHEL 6.7 or higher 64-bit
Product versions	IBM Spectrum Symphony 7.1.2 and IBM Spectrum Conductor with Spark 2.2.0

Installation

Follow the instructions in this section to download and install this enhancement on Linux hosts in your cluster.

Prerequisites

IBM Spectrum Symphony 7.1.2 and IBM Spectrum Conductor with Spark 2.2.0 must be installed in your cluster.

Packages

Name	**Description**
`sym-7.1.2.0_x86_64_cws-2.2.0.0_x86_64_build450098-sample.tar.gz`	Package containing the sample plug-in. It's ready to install for demo purpose.
`sym-7.1.2.0_x86_64_cws-2.2.0.0_x86_64_build450098.tar.gz`	Package containing the source code(.c), the header source file (.h), the Makefile, and the library (.a) for building a custom plug-in.
`Readme_build450098.pdf`	Readme file.

Developing, building and installing the custom plug-in

To implement your own logic for authentication to the cluster, build a custom plug-in with your own authentication logic. You must first implement the three authentication APIs provided by this enhancement, then build the custom plug-in.

Custom plug-in development

API description

The following table provides description for the three APIs exposed for customization:

API	Input	Output	Return Value	Additional Notes
int customized_initialize()	N/A	N/A	Initialization result. Valid values are **SECE_OK** and **SECE_FAIL**.	User can parse the plug-in's

			If initialization succeeds, return SECE_OK; otherwise, return SECE_FAIL.	configuration file inside this API; they can define the parameter configuration format. They can also do other initialization related to their authentication. Note: The sample will give a method for parsing the parameter configured with format "key=value"
int customized_auth(char *username, char *password)	username and password	N/A	Authentication result. Valid values are **AUTH_PASS**, **AUTH_ERROR**,and **AUTH_CONTINUE_DEFAULT**. If authentication succeeds, return AUTH_PASS; if authentication fails, return AUTH_ERROR; if continue to do authentication against EGO database, return AUTH_CONTINUE_DEFAULT.	With this API, user can implement their own logic to authenticate the passed-in username and password.
int customized_finalize()	N/A	N/A	Finalization result. Valid values are **SECE_OK** and **SECE_FAIL**. If the finalization succeeds, return SECE_OK; otherwise, return SECE_FAIL.	With this API, user can do their own finalization logic before server exits. For example: free memory. The outside logic will not take any action except log a message if this step fails.

Building the custom plug-in

a. Copy the `sym-7.1.2.0_x86_64_cws-2.2.0.0_x86_64_build450098.tar.gz` file to your build host and decompress the package to a directory, which is hereafter referred to as the "extract directory".

   You should see the following files and folders in the extract directory:

   - `sec_ego_ext_plugin.a`: Static library used for building the custom plug-in.
   - `sec.h`: Header file to be included by the customized source code file.
   - `sample/sec_customize_auth.c`: Sample source code file.

- `sample/Makefile`: Sample make file.

b. Place the customized source code file and make file in the extract directory and build the custom plug-in.

The following steps how to build sample plug-in; use these steps as a reference to build your custom plug-in:

- Copy the sample source code file and sample make file from the subdirectory "`sample/`" to the extract directory.
- Edit the Makefile in the extract directory and set the GCC value to the full path of your GCC. For example:

```
# Define the value of GCC to your own gcc full path, use
GCC4.8.2

GCC=/usr/bin/gcc
```

- In the extract directory, run the "`make`" command to build the plug-in:

```
make
```

You have now built the plug-in, named `sec_ego_ext_custom.so`.

c. Ensure that file ownership for `sec_ego_ext_custom.so` is set to the cluster administrator account and file permissions are set to 644.

Installing the custom plug-in

a. Log on to the master host as the cluster administrator, disable Symphony applications, and shut down the cluster:

```
$ soamcontrol app disable all

$ egosh service stop all

$ egosh ego shutdown all
```

b. Log on to all management hosts as the cluster administrator and copy the custom plug-in that you built previously (`sec_ego_ext_custom.so`) to the `$EGO_LIBDIR` directory.

c. Configure the custom plug-in as the authentication plug-in for your cluster as described in the "Configuration and usage" section.

d. Start your cluster and enable Symphony applications:

```
$ egosh ego start all

$ soamcontrol app enable <appName>
```

Installing the sample plug-in

This enhancement provides a sample plug-in that is built with sample authentication logic. This sample plug-in demonstrates usage of the three authentication APIs and can be used for testing purposes.

Description for the authentication logic in the sample plug-in:
The authentication will always succeed for the user that has been defined as "pass" in the configuration file `customauth.conf`(see "`Configuration and usage`" section):
The authentication will always fail for the user that has been defined as "fail";
The authentication will continue with EGO authentication for the user that wasn't defined or defined with another value in the configuration file.

Follow these steps to install the sample plug-in that you can use for authentication to your cluster:

a. Log on to the master host as the cluster administrator, disable Symphony applications, and shut down the cluster:

```
$ soamcontrol app disable all
$ egosh service stop all
$ egosh ego shutdown all
```

b. Copy the `sym-7.1.2.0_x86_64_cws-2.2.0.0_x86_64_build450098-sample.tar.gz` file to the `$EGO_TOP` directory on all management hosts, and decompress the package.

c. Configure the sample plug-in as the authentication plug-in for your cluster as described in the "Configuration and usage" section.

d. Start your cluster and enable Symphony applications:

```
$ egosh ego start all
$ soamcontrol app enable <appName>
```

Uninstalling the plug-in

If required, follow these steps to remove the custom or sample plug-in as the authentication plug-in for your cluster:

1. Log on to the master host as the cluster administrator, disable Symphony applications, and shut down the cluster:

```
$ soamcontrol app disable all
$ egosh service stop all
$ egosh ego shutdown all
```

2. On all hosts, recover the `ego.conf` file that you previously backed up in the "Configuration and usage" section.
3. Start your cluster and enable Symphony applications:

```
$ egosh ego start all
$ soamcontrol app enable <appName>
```

Configuration and usage

Configuring the authentication plug-in on management hosts

1. On each management host, back up the `$EGO_CONFDIR/ego.conf` file.
2. Edit the following parameters in the `$EGO_CONFDIR/ego.conf` file:

   - **`EGO_SEC_PLUGIN`**: Specify the name of the authentication plug-in (`sec_ego_ext_custom`):

     ```
     EGO_SEC_PLUGIN=sec_ego_ext_custom
     ```

   - **`EGO_SEC_CONF`**: Specify the plug-in configuration in the format "`path_to_plugin_conf_dir,created_ttl,plugin_log_level,path_to_plugin_log_dir`", where:
     - o `path_to_plugin_conf_dir`(required): Specifies the absolute path to `$EGO_CONFDIR`, where the plug-in configuration file is located. See **step 2** for details on creating the configuration file.
     - o `created_ttl`(optional): Specifies a time-to-live duration for the authentication token

sent from the client to the server. Valid values are 0 or empty (indicating that the default value of 10 hours must be used).

- `plugin_log_level`(optional): Specifies the log level for the plug-in. Valid values are DEBUG, INFO, WARN, and ERROR. As a best practice, set the log level as ERROR or WARN. A lower level causes too many messages to be logged, making it harder to troubleshoot if required.
- `path_to_plugin_log_dir`(optional): Specifies the absolute path to the directory where the plug-in's logs are located.

For example:

```
EGO_SEC_CONF="/opt/egoshare/kernel/conf,0,ERROR,/opt/cluster/MH/kernel/
log"
```

3. Define a new configuration file for the plug-in, create the file under `$EGO_CONFDIR` on all management hosts and configure its parameters.

   For example, it's required to finish the step below for the sample plug-in.
   Create the `customauth.conf` file under `$EGO_CONFDIR` and configure users in the file, for example, define two test users to demonstrate the enhancement via the sample plug-in as follows:
   ```
   test_user1=pass
   test_user2=fail
   ```
   We will use the these test users in the **"Verifying authentication through the sample plug-in" section".**

Configuring the authentication plug-in on compute and client hosts

1. Back up the `ego.conf` file, which is located on all compute hosts at `$EGO_CONFDIR` and on all client hosts at `$SOAM_HOME/conf/`.
2. Modify the **`EGO_SEC_PLUGIN`** parameter in the `$EGO_CONFDIR/ego.conf` file on all compute hosts or `$SOAM_HOME/conf/ego.conf` file on all client hosts as follows:
   ```
   EGO_SEC_PLUGIN=sec_ego_ext_co
   ```

Verifying authentication through the sample plug-in

If you used the sample plug-in, follow these steps to verify authentication to your cluster assuming the `test_user1` and `test_user2` are defined in `$EGO_CONFDIR/customauth.conf:`

1. Log on to the cluster as the "Admin" cluster administrator. For example:
   ```
   $ egosh user logon -u Admin -x Admin
     Logged on successfully
   ```
2. Add three test users to the EGO database and assign the "CLUSTER_READONLY_ADMIN" role for these users. For example:
   ```
   $ egosh user add -u test_user1 -x 1
     User account <test_user1> added successfully
   $ egosh user add -u test_user2 -x 2
     User account <test_user2> added successfully
   $ egosh user add -u test_user3 -x 3
     User account <test_user3> added successfully
   ```

```
$ egosh user assignrole -u test_user1 -r CLUSTER_READONLY_ADMIN
  Role <Cluster Admin (Read only)> is assigned to user <test_user1>.
$ egosh user assignrole -u test_user2 -r CLUSTER_READONLY_ADMIN
  Role <Cluster Admin (Read only)> is assigned to user <test_user2>.
$ egosh user assignrole -u test_user3 -r CLUSTER_READONLY_ADMIN
  Role <Cluster Admin (Read only)> is assigned to user <test_user3>.
```

3. Log on as user "test_user1" with any password, then run some commands to verify. For example:

   ```
   $ egosh user logon -u test_user1 -x randompass
     Logged on successfully
   $ egosh rg
   $ soamview app
   ```

   Authentication must succeed.

4. Log on as user "test_user2" with any password. For example:

   ```
   $ egosh user logon -u test_user2 -x 2
     Cannot logon. Authentication failed.
   ```

   Authentication must fail.

   Find the ERROR log in the plug-in's server log `ego_ext_plugin_server.log` in the folder specified by the `EGO_SEC_CONF` parameter. For example:

   ```
   Thu Apr 13 16:32:46 2017 ERROR [1083608] server_start(): The
   customized_auth() function returns AUTH_ERROR. Check the custom
   authentication log for detailed reason.

   Thu Apr 13 16:32:46 2017 ERROR [1083608] server_start(): Auth failed,
   **out=F
   ```

5. Log on as user "test_user3" with its password "3", authentication must succeed. With any other password, authentication must fail. For example:

   ```
   $ egosh user logon -u test_user3 -x 3
     Logged on successfully
   $ egosh user logon -u test_user3 -x reandompass
     Cannot logon. Authentication failed.
   ```

   Find messages in the plug-in's server log `ego_ext_plugin_server.log` (DEBUG level configured). For example:

   ```
   Thu Apr 13 17:36:13 2017 DEBUG [1137064] customized_auth(): Did not find
   user test_user3 in the customauth.conf file, will continue with EGO
   authentication.

   Thu Apr 13 17:36:13 2017 DEBUG [1137064] server_start(): Continue
   authentication against EGO database.

   Thu Apr 13 17:36:13 2017 DEBUG [1137064] checkUserPasswordDefault():
   entering...

   Thu Apr 13 17:36:13 2017 DEBUG [1137064] the directory is:
   /opt/xjli/cluster/MH/kernel/conf/users.xml

   Thu Apr 13 17:36:13 2017 ERROR [1137064] server_start(): Auth failed,
   ```

```
**out=F
```

6. Log on as EGO default users (for example: “Admin” and “Guest”) from the cluster management console or the command line. Authentication must work as before.

   NOTE: Before logging in to the cluster management console, clear the browser cache.

Verifying authentication through the custom plug-in

If you used the custom plug-in, follow these steps to verify authentication to your cluster:

1. Log on to the cluster as the “Admin” cluster administrator. For example:

   ```
   $ egosh user logon -u Admin -x Admin
   ```

2. Add the users that must be authenticated with the custom logic to the EGO database (from the GUI or from the command line using the “egosh user add” command),and assign “CONSUMER_ADMIN” role for the user. For example:

   ```
   $ egosh user add -u test_user_abc -x pswdabc123
   ```

   ```
   $ egosh user assignrole -u test_user_abc -r CONSUMER_ADMIN -p /
   ```

   NOTE: If the user won’t be authenticated against the EGO database, you can use any random string as the password.

3. Test access for the users by logging in to the cluster as those users from the cluster management console or the command line.

Troubleshooting

If you encounter authentication issues, check the logs for errors. All authentication errors are logged to the plug-in’s server log `ego_ext_plugin_server.log` in the log directory, which is specified by the `EGO_SEC_CONF` parameter in the `$EGO_CONFDIR/ego.conf` on management hosts.

If configuration errors or the plug-in’s own issue cause the server to not load the plug-in successfully, look for ERROR logs in the authentication server logs (such as the VEMKD logs).

For example, when the plug-in file does not exist under `$EGO_LIBDIR`, the cluster cannot start successfully and ERROR messages in the VEMKD log:

```
2017-04-13 17:00:01.000 CST ERROR [746373:139724319766304] secinit: dlerror():
/opt/xjli/cluster/MH/3.5/linux-x86_64/lib/sec_ego_ext_custom.so: cannot open
shared object file: No such file or directory.
```

```
2017-04-13 17:00:01.000 CST ERROR [746373:139724319766304] secinit: failed to
initialize security module.
```

Copyright and trademark information