Authenticating External Configuration Files

Authenticating External Configuration Files

Prior to Cisco ESC Release 4.0, ESC supports several external configuration files and scripts as part of day 0 configuration, monitoring, deployment and LCS actions. ESC supports getting these files from a remote server with or without authentication as part of the deployment.

Starting from ESC Release 4.0, the file locator attribute is defined at the deployment level, that is, directly under the deployment container. This allows multiple VM groups and their day 0 configuration and LCS actions to reference the same file locator wherever needed within the deployment.

Sample deployment data model is as follows:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>sample-tenant</name>
      <deployments>
        <deployment>
          <name>sample-deployment</name>
          <file_locators>
            <file_locator>
              <name>post_deploy_alive_script</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>/share/qatest/vnfupgrade/lcspostdeployalive.sh</remote_path>
                <local_target>vnfupgrade/lcspostdepalive.sh</local_target>
                <persistence>FETCH_ALWAYS</persistence>
                <properties/>
              </remote_file>
            </file_locator>
            <file_locator>
              <name>asa-day0-config</name>
              <remote_file>
                <file_server_id>http-my-server</file_server_id>
                <remote_path>/share/qatest/day0/asa_config.sh</remote_path>
                <local_target>day0.1/asa_config.sh</local_target>
                <persistence>FETCH_ALWAYS</persistence>
              </remote_file>
            </file_locator>
```

Authenticating External Configuration Files

```
      <file_locator>
        <name>scriptlocator</name>
        <remote_file>
          <file_server_id>dev_test_server</file_server_id>
          <remote_path>/share/users/gomoore/actionScript.sh</remote_path>
          <local_target>action/actionScript.sh</local_target>
          <persistence>FETCH_MISSING</persistence>
          <properties/>
        </remote_file>
      </file_locator>
    </file_locators>
    <policies>
      <policy>
        <name>VNFUPGRADE_POST_DEPLOY_ALIVE</name>
        <conditions>
          <condition>
            <name>LCS::POST_DEPLOY_ALIVE</name>
          </condition>
        </conditions>
        <actions>
          <action>
            <name>post_deploy_alive_action</name>
            <type>SCRIPT</type>
            <properties>
              <property>
                <name>file_locator_name</name>
                <value>post_deploy_alive_script</value>
              </property>
            </properties>
          </action>
        </actions>
      </policy>
    </policies>
  <vm_group>
      <name>ASA-group</name>
      <image>ASAImage</image>
      <flavor>m1.large</flavor>
      <recovery_policy>
        <max_retries>1</max_retries>
      </recovery_policy>
      <scaling>
        <min_active>1</min_active>
        <max_active>1</max_active>
        <elastic>true</elastic>
      </scaling>
      <placement>
        <type>affinity</type>
        <enforcement>strict</enforcement>
      </placement>
      <bootup_time>120</bootup_time>
      <recovery_wait_time>60</recovery_wait_time>
      <interfaces>
        <interface>
          <nicid>0</nicid>
          <network>my-net</network>
        </interface>
      </interfaces>
      <kpi_data>
        <kpi>
          <event_name>VM_ALIVE</event_name>
          <metric_value>1</metric_value>
          <metric_cond>GT</metric_cond>
          <metric_type>UINT32</metric_type>
          <metric_occurrences_true>1</metric_occurrences_true>
```

```
                <metric_occurrences_false>5</metric_occurrences_false>
                <metric_collector>
                  <nicid>0</nicid>
                  <type>ICMPPing</type>
                  <poll_frequency>5</poll_frequency>
                  <polling_unit>seconds</polling_unit>
                  <continuous_alarm>false</continuous_alarm>
                </metric_collector>
              </kpi>
            </kpi_data>
            <rules>
          <admin_rules>
                  <rule>
                    <event_name>VM_ALIVE</event_name>
                    <action>ALWAYS log</action>
                    <action>TRUE servicebooted.sh</action>
                    <action>FALSE recover autohealing</action>
                  </rule>
                </admin_rules>
            </rules>
            <config_data>
              <configuration>
                <dst>ASA.static.txt</dst>
                <file_locator_name>asa-day0-config</file_locator_name>
              </configuration>
            </config_data>
            <policies>
              <policy>
                <name>SVU1</name>
                <conditions>
                <condition><name>LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH</name></condition>

                </conditions>
                <actions>
                  <action>
                    <name>LOG</name><type>pre_defined</type>
                  </action>
                  <action>
                    <name>pre_vol_detach</name>
                    <type>SCRIPT</type>
                    <properties>
                      <property>
                        <name>file_locator_name</name>
                        <value>scriptlocator</value>
                      </property>
                      <property>
                        <name>exit_val</name>
                        <value>0</value>
                      </property>
                    </properties>
                  </action>
                </actions>
              </policy>
            </policies>
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

You must configure a remote server (file server) separately using the APIs before performing any deployment. Both REST and NETCONF APIs are supported

- A remote server with URL, authentication details including username, and password. You can either use REST or NETCONF to configure.

  Note The username and password are optional. The password is encrypted within ESC.

  You must configure the remote file server before deployment. You can update the credentials anytime during the deployment.

- File locator is added to the deployment data model. It contains a reference to the file server, and the relative path to the file to be downloaded.

To get files remotely with authentication, you must

1. Add a remote server.
2. Refer the remote server in the file locator. The file locator is part of config data in day 0 and LCS action blocks.
3. The day 0 and lifecycle stage (LCS) scripts will then be retrieved based on the file locator as part of the deployment.

The file server parameters include:

- id—used as the key and identifier for a file server.
- base_url—the address of the server. (e.g. http://www.cisco.com or https://192.168.10.23)
- file_server_user—the username to use when authenticating to the server.
- file_server_password—string containing the password for authenticating to the server. Initially the user provides a cleartext string, which is encrypted internally.
- properties—name-value pair for extensibility in the future.

The file locator parameters include:

- name—used as the key and identifier for a file locator.
- local_file or remote_file—choice of file location. Local file is used to specify a file existing on the ESC VM file system already. The remote_file is used to specify a file to fetch from a remote server.
  - file_server_id—id of the File Server object to fetch the file from.
  - remote_path—path of the file from the base_url defined in the file server object.
  - local_target—optional local relative directory to save the file.
  - properties—name-value pairs of of information that may be required.
  - persistence—options for file storage. Values include CACHE, FETCH_ALWAYS and FETCH_MISSING (default).
- checksum—optional BSD style checksum value to use to validate the transferred file's validity.

The file server values such as server connectivity, file existence, checksum and so on will be verified for validity.

The encrypted_data values in the file_server_password and properties encrypted_data fields are encrypted using AES/128bits in CFB mode for transmission.The data remains encrypted until it is required for accessing the server. For more information on encrypted values, see Encrypting Configuration Data.

Example of file servers,

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <file_servers>
    <file_server>
      <id>server-1</id> <!-- unique name for server -->
      <base_url>https://www.some.server.com</base_url>
      <file_server_user>user1</file_server_user>
      <file_server_password>sample_password</file_server_password>
      <!-- encrypted value -->
      <!-- properties list containing additional items in the future -->
      <properties>
        <property>
          <name>server_timeout</name>
          <value>60</value>
      <!-- timeout value in seconds, can be over-ridden in a file_locator -->
        </property>
      </properties>
    </file_server>
    <file_server>
      <id>server-2</id>
      <base_url>https://www.some.other.server.com</base_url>
          <properties>
        <property>
          <name>option1</name>
          <encrypted_value>$8$EADFAQE</encrypted_value>
        </property>
    </file_server>
  </file_servers>
</esc_datamodel>
```

Example for day 0 configuration

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants><tenant>
      <name>sample-tenant</name>
      <deployments><deployment>
          <name>sample-deployment</name>
          <vm_group>
            <name>sample-vm-group</name>
            <config_data>
              <!-- exisiting configuration example - remains valid -->
              <configuration>
                <file>file:///cisco/config.sh</file>
                <dst>config.sh</dst>
              </configuration>
              <!-- new configuration including use of file locators -->
              <configuration>
                <dst>something</dst>
                <file_locators>
                  <file_locator>
                    <name>configlocator-1</name> <!-- unique name -->
                    <remote_file>
                      <file_server_id>server-1</file_server_id>
                      <remote_path>/share/users/configureScript.sh</remote_path>
                      <!-- optional user specified local silo directory -->
                      <local_target>day0/configureScript.sh</local_target>
```

```
                      <!-- persistence is an optional parameter -->
                      <persistence>FETCH_ALWAYS</persistence>
                      <!-- properties in the file_locator are only used for
                           fetching the file not for running scripts -->
                      <properties>
                        <property>
                          <!-- the property name "configuration_file" with value "true"
indictates this is the
                               script to be used just as using the <file> member case of
the configuration -->
                          <name>configuration_file</name>
                          <value>true</value>
                        </property>
                        <property>
                          <name>server_timeout</name>
                          <value>120</value> <!-- timeout value in seconds, overrides the
file_server property -->
                        </property>
                        </properties>
                      </remote_file>
                      <!-- checksum is an optional parameter.
                           The following algorithms are supported: SHA-1, SHA-224, SHA-256,
SHA-384, SHA-512 -->
                      <checksum>SHA256 (configureScript.sh) =
dd526bb2c0711238ec2649c4b91598fb9a6cf1d2cb8559c337c5f3dd5ea1769e</checksum>
                    </file_locator>
                    <file_locator>
                      <name>configlocator-2</name>
                      <remote_file>
                        <file_server_id>server-2</file_server_id>
                        <remote_path>/secure/requiredData.txt</remote_path>
                        <local_target>day0/requiredData.txt</local_target>
                        <persistence>FETCH_ALWAYS</persistence>
                        <properties/>
                      </remote_file>
                    </file_locator>
                  </file_locators>
                </configuration>
              </config_data>
            </vm_group>
          </deployment></deployments>
      </tenant></tenants>
</esc_datamodel>
```

For more details on day 0 configuration and LCS actions, see day 0 configuration, and Redeployment Policy sections.

Encrypting Configuration Data

You can encrypt configuration data with secret keys and private information. In ESC, the day 0 configuration, day 0 configuration variables, VIM connector and VIM user, and LCS actions contain secret keys.

ConfD provides encrypted string types. Using the built-in string types, the encrypted values are stored in ConfD. The keys used to encrypt the values are stored in confd.conf.

Encrypting data is optional. You can use the encrypt_data value to store data if necessary.

In the example below, the day 0 configuration data has encrypted values. The encrypted_data uses the built in string type *tailf:aes-cfb-128-encrypted-string*.

```
choice input_method {
  case file {
    leaf file {
      type ietf-inet-types:uri;
    }
  }
  case data {
    leaf data {
      type types:escbigdata;
    }
  }
  case encrypted_data {
    leaf encrypted_data {
      type tailf:aes-cfb-128-encrypted-string;
    }
  }
}
```

Generating Advanced Encryption Standard (AES) Key

The AES key is 16 bytes in length, and contains a 32 character hexadecimal string.

You must configure the AES key in confd.conf for the encryption to work.

```
/opt/cisco/esc/esc-confd/esc_production_confd.conf
```

```
<encryptedStrings>
    <AESCFB128>
        <key>0123456789abcdef0123456789abcdef</key>
        <initVector>0123456789abcdef0123456789abcdef</initVector>
    </AESCFB128>
</encryptedStrings>
```

A default AES key is available in confD:

```
0123456789abcdef0123456789abcdef
```

The confD key is hard-coded. The escadm.py generates a random AES key and replaces the default confD AES key before confD starts.

Encrypting Variables

You can encrypt variables such as passwords and chassis ids within the day 0 configuration using *encrypted_val*. ESC allows you to choose either *val* or *encrypted_val* for variables within the deployment datamodel.

The text within the *encrypted_val* is encrypted into the confD Database (CDB) and PostgreSQL DB. The text is decrypted only at the point of use (not when the data is at rest). In the ESC logs, the text in *encrypted_val* is masked.

In the example below, the northbound client (Netconf or REST) populates the *encrypted_val* with plain text. When the deployment request is processed by ESC ConfD, the plain text is encrypted into the ESC databases.

```
<config_data>
  <configuration>
    <dst>vnf_day0.cfg</dst>
    <data>file://opt/cisco/esc/esc_database/vnf_day0.cfg</file>
    <variable>
      <name>user</name>
      <val>admin</val>
    </variable>
    <variable>
      <name>password</name>
```

```
      <encrypted_val>ADMIN-PASSWORD</encrypted_val>
    </variable>
```

When the *encrypted_val* is retrieved from ConfD configuration via netconf or CLI, it displays the plain text in the encrypted form.

```
<config_data>
  <configuration>
    <dst>vnf_day0.cfg</dst>
    <data>file://opt/cisco/esc/esc_database/vnf_day0.cfg</file>
    <variable>
      <name>user</name>
      <val>admin</val>
    </variable>
    <variable>
      <name>password</name>

<encrypted_val>$8$cVl6r9aR7W3wmHLYUrAOQHnjJGH0XltJjiCBTXANJFV0sJfb/NF+lEJiUA0j/JxA</encrypted_val>

    </variable>
```

Note A single value is stored in *encrypted_val*. The same variable value is substituted into the day 0 configuration template for all VMs in the scale group.

You can use *encrypted_val* in the day 0 configuration to secure the chassis id. The value of the chassis id is provided by the northbound client or operator performing the VNF upgrade (chassis id generated through the script during VNF deployment is not supported).

```
<config_data>
  <configuration>
    <dst>staros_param.cfg</dst>
    <file>file://opt/cisco/esc/images/staros_param_upf.cfg</file>
    <variable>
      <name>CHASSIS_ID</name>
      <encrypted_val>VALUE-PROVIDED-BY-NORTHBOUND-OPERATOR</encrypted_val>
    </variable>
```

For information on day 0 configuration, see Day Zero Configuration.

Cisco Elastic Controller Services Script for Encoding ConfD AES Encrypted Strings

This feature provides scripts to encode the AES encrypted strings compatible for use in config requests, for example, dep.xml. Following are the two scripts (alternatives) that provide the same function:

- `esc_nc_cli encrypt`
- `esc_confd_encrypt` - It is a standalone script to use on the ESC VM or a remote Linux server with connectivity to the ESC VM.

The following commands help you to encrypt the plain text into AES encrypted string.

```
esc_nc_cli encrypt
```

For example:

```
admin@esc-01$ esc_nc_cli encrypt
Enter plain text (input is not echoed to terminal) >  *********
admin@127.0.0.1's password:
$8$aaCBcnVmZ+6lEV1FvhhitzQMLisLc3pxk1uUh+7DL4A=


admin@esc-01$ esc_nc_cli encrypt input.txt
admin@127.0.0.1's password:
$8$SLwFZuA0m0Rgf69fPNOeiq4ispm5H1SZIVGzzDd5R2g=
```

The following command is equivalent to `esc_nc_cli`, implemented as a separate, standalone script.

For example:

```
admin@esc-01$ esc_confd_encrypt
Enter plain text (input is not echoed to terminal) > ********
admin@localhost's password:
$8$QL5vFU1vt3KEs3kKIrC0+Faq8cF83WdptPO45GTIBGA=

admin@esc-01$ esc_confd_encrypt --file input.txt
admin@localhost's password:
$8$uzN7+kMgCf4RLxB5R0qMnLIbixO6EUpliUuHJRwR944=
```

The following command connects to ConfD CLI ssh (port 2024).

For example:

```
admin@esc-01$ esc_nc_cli cli
ssh -o StrictHostKeyChecking=no -p 2024 admin@127.0.0.1
admin@127.0.0.1's password: *****

admin connected from 127.0.0.1 using ssh on esc-01
admin@esc-01>
```

Using the Scripts from a Remote Host

You can use both the scripts to perform the encryption at a remote ESC. For example, a linux server with connectivity to the ESC VM, northbound client or administrative 'jump host'.

For example:

```
abc@my-server-39:~$ esc_confd_encrypt --host 172.25.0.89 --user admin
Enter plain text (input is not echoed to terminal) >
admin@172.25.0.89's password:
$8$VUnQkT30fKqAWWCiyDPkqUjS+jDd0/sNIyGNd4bVppE=

abc@my-server-39:~$ esc_nc_cli encrypt --host 172.25.0.89 --user admin
Enter plain text (input is not echoed to terminal) >
admin@172.25.0.89's password:
$8$uRBKqpZZ9rcUIrfBam0WfCXq3tirTD+FRcafBqAArRs=

abc@my-server-39:~$ esc_nc_cli encrypt --host 172.25.0.89 --user admin --password 'REDACTED'
Enter plain text (input is not echoed to terminal) >
$8$iG9vvLAqk69wUSMVMVf5XDpwkdDi/P1V9ucJlXKn2NQ=
```

Enabling Password-less Access to the Scripts with Public Key Authentication

There are two methods to enable password-less (public key authentication) to ConfD (netconf and ssh cli) for direct use of through wrapper utilities, for examples `esc_nc_cli` and `esc_confd_encrypt`.

Following is an example for creating a private key pair and config public key auth in ConfD (preferred):

```
admin@esc-01$ ssh-keygen -t rsa -b 2048 -C "admin" -N "" -f ~/.ssh/test_confd_rsa
Generating public/private rsa key pair.
Your identification has been saved in /home/admin/.ssh/test_confd_rsa.
Your public key has been saved in /home/admin/.ssh/test_confd_rsa.pub.
The key fingerprint is:
SHA256:u3/dpc4iY6/60fiGjGeJjMcigUKlSrxCptZWYo8JQ6o admin
The key's randomart image is:
+---[RSA 2048]----+
|                 |
| . .             |
|+ o              |
|.X o .           |
|O *.*   S        |
|Eo.=..   . o    .|
|o..  . +.+oo.. o.|
|    . o *.Xo+.o .|
|     . ooB+Booo  |
+----[SHA256]-----+
admin@esc-01$ sudo mkdir --mode=700 -p /var/confd/homes/admin/.ssh
admin@esc-01$ sudo cp ~/.ssh/test_confd_rsa.pub /var/confd/homes/admin/.ssh/authorized_keys
admin@esc-01$ sudo chown -R esc-user:esc-user /var/confd/homes/admin/.ssh

admin@esc-01$ printf "value-of-encrypted_val" | esc_nc_cli encrypt --privKeyFile
~/.ssh/test_confd_rsa
$8$VmDBKYupSGUCaILw8g2VYykVD9D16jA44sQNglFUUAv+uQtO0BmEtSC85vfuRJu0

admin@esc-01$ printf "value-of-encrypted_val" | esc_confd_encrypt --privKeyFile
~/.ssh/test_confd_rsa
$8$oFXwX1jeIHVxmBuMdPe6Vz6usaSahPVh0gZEGHm0uoAvK+twC0kUK5w7/QY0goUM

admin@esc-01$ cat .ssh/config
Host localhost 127.0.0.1
   Port 2024
   IdentityFile ~/.ssh/test_confd_rsa

admin@esc-01$ printf "value-of-encrypted_val" | esc_nc_cli encrypt
$8$GZ4+2nSo/YklKVk8RTdNR9oDJjWe89VsUiUR2FnIwtW4WPSXLivOXbmZnHR2YpfP

admin@esc-01$ printf "value-of-encrypted_val" | esc_confd_encrypt
$8$ggQaMq3QEIhS+1P8gmtr47LwdPyrCFoHHC2jzv2vKnxBFvIPNQapHurj+bcHfpEe
```

Following is the example for enabling ConfD keys to access ConfD with built-in esc-nc-admin account (offered for backwards compatibility):

```
admin@esc-01$ sudo escadm confd keygen --user admin
Generated SSH key pair for user admin and authorized them for user esc-nc-admin

admin@esc-01$ printf "value-of-encrypted_val" | esc_nc_cli encrypt
$8$4c5m8cqK21VNyb1gCfc77p41LKxA9Ar8n6CApQwNst8yk/ilDphiDXetmHPmKuvP

admin@esc-01$ printf "value-of-encrypted_val" | esc_confd_encrypt
$8$yY8sG6leUkrnY+fBUrYVmnwPSBY9aIrUKXmpaHVGfvNWggLuSPkqZcRCjejPej+y
```