Videoscape Control Suite SASL API Guide

Overview

Introduction

The Cisco Videoscape Control Suite (VCS) Message Infrastructure provides a Simple Authentication and Security Layer(SASL) component that allows integrating plugins created by third-parties into the VCS system. This document describes the SASL APIs supported by VCS Versions 2.5 and 3.0.

Audience

This document is written for system operators. Our engineers may also find this document to be useful.

Users of this guide should have a basic understanding of the C++ programming language, as well as having Linux developing and debugging skills, to create new SASL plugins with the Cisco Conductor SDK APIs. Any project building tool (like GNU Automake) can be used to build the project.

Document Version

This is the first formal release of this document.

In This Document

SASL APIs

This section provides details and usage for the SASL APIs.

Mechanism Initialization

- Implement **getMechanismName()** to return the name of a mechanism to the SASL component.

  Note: A mechanism name must be unique in the SASL mechanism list.

  Example:

```
/* static */
const char* SampleMech::name = "SAMPLE";

const char* SampleMech::getMechanismName()
const
{
    return name;
}
```

- Implement the **MechanismInitFunc** function as defined for the Mechanism.hpp API, where the entry point SASL component loads each plugin.

  Example:

```
typedef MechanismFactory
(*MechanismInitFunc)(SaslContext&, std::string&);
```

  Notes:

  - SaslContext represents the SASL component instance. The mechanism must return the name of the plugin and the MechanismFactory function.

    Example:

```
using namespace sasl_auth;

MechanismFactory sasl_sample(SaslContext
&context, string &name)
{
name = SampleMech::name;
return &SampleMech::factory;
}
```

  - The name returned from the **MechanismInitFunc** function must be identical as the name entered from the GUI management interface. Otherwise, the SASL component will log an error and ignore the plugin as a security issue. You must return the correct factory function entry in the initialization function. Otherwise, the behavior of the SASL component could be abnormal.

- A **MechanismFactory** is the factory function to create the new mechanism instance when a new client initiates the authentication request. The SASL component calls this function to get a new Mechanism instance to handle the authentication request in a separate working thread. You must return either a new instance of a mechanism or a NULL value. Otherwise, the behavior of the SASL component could be abnormal.

  Example:

```
typedef Mechanism*
(*MechanismFactory)(SaslContext&);

/* static */ Mechanism*
SampleMech::factory(SaslContext &context)
{
return new SampleMech(context);
}
```

Mechanism Authentication

The main authentication functions are **start ()** and **step ()**, derived from class Mechanism.

- **start()** is called when the client has selected the mechanism and initiated a new SASL negotiation session.
- **step()** is called when the client sends a response to the previous challenge. If multiple challenge/response steps are required, step() will be called multiple times.

Example:

```
Bool start(const std::string &auth, const
std::string &domain, SaslErrorCode &error,
std::string &challenge);

bool step(const std::string &response, const
std::string &domain, SaslErrorCode &error,
std::string &challenge);
```

The following parameters are passed to the start() function from the SASL component.

Parameter	Description
auth	The content of the initial <auth/> element from the client side. If the following stanza is received from an XMPP client, the auth string will be **AHRlc3QxAHRlc3Q=**: **<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='PLAIN'> AHRlc3QxAHRlc3Q=** **</auth>**
domain	The user domain sent from the client
error	The error code returned by the mechanism after handling the auth request. The error code has been predefined in SaslErrorCode.
challenge	The new challenge initiated from the mechanism. The SASL component transmits the challenge to the client side. For example, if the mechanism returns a string "AABBCCDDEEFF," the SASL component sends the following XMPP stanza to the client: **<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>** **AABBCCDDEEFF** **</challenge>**
return value	A boolean value indicates whether the mechanism requires the next challenge/response iteration. ■ **True**—indicates that the initial <auth/> contains no error and the mechanism will initiate a new challenge to the client to continue the authentication process. ■ **False**—indicates that the mechanism will end the authentication session with either eSuccess or an error code predefined in SaslErrorCode.

When a mechanism returns "true" from the start () function, the SASL component returns the XMPP stanza to the client and expects the client to respond with the challenge within a limited duration (default is 30 seconds). When the client sends the response within this duration, the step() function is called with the response payload. If the client does not respond back within this duration, the SASL component cancels the session and destroys the mechanism instance.

The following parameters are passed to the step() function from the SASL component.

Parameter	Description
response	The content of the <response/> element from the client side. In the following example, the response string will be "AABBCCDDEEFF." **<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>** **AABBCCDDEEFF** **</response>**
domain	The user domain sent from the client
error	The error code returned by the mechanism after handling the response. The error code has been predefined in SaslErrorCode.
challenge	The new challenge initiated from the mechanism. The SASL component transmits the challenge to the client side. For example, if the mechanism returns a string 'AABBCCDDEEFF', the SASL component sends the following XMPP stanza to the client: **<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>** **AABBCCDDEEFF** **</challenge>**
return value	A boolean value indicates whether the mechanism requires the next challenge/response iteration. ■ **True**—indicates that the initial <auth/> contains no error and the mechanism will initiate a new challenge to the client to continue the authentication process. ■ **False**—indicates the mechanism will end the authentication session with either eSuccess or an error code predefined in SaslErrorCode.

After the client is successfully authenticated, the mechanism must return the valid user JID by setting the _username variable. The SASL component calls getUserName() to create the JID (username@domain) by _username and domain. The domain is the second parameter in the start ()/step () function.

Example:

```
bool start(const std::string &auth, const std::string
&domain, SaslErrorCode &error, std::string &challenge)
{
...
_username = user; // set username to create
JID(user@domain)
error = eSuccess; // indicate the success of
authentication
return false; //no need further challenge/response
iteration
}
```

Mechanism Utilities

The SaslContext class is the abstract base class that is derived by the SASL component. The APIs described in this section allow you to get properties from the SASL component to complete the authentication process.

Example:

```
enum SaslErrorCode getUsersPass(const std::string
&user, const std::string &domain, std::string
&pass);
bool getAuthzEnabled();
void saslLog(LogLevel level, const char* channel,
const char* message);
```

getUsersPass

Example:

```
enum SaslErrorCode getUsersPass(const std::string
&user, const std::string &domain, std::string
&pass);
```

The getUsersPass function queries the VCS database to get the password string by user name and domain name. The following error codes can be returned from the function.

Parameter	Possible reason
eNotAuthorized	This account (user@domain) does not exist in the database.
eAccountDisabled	This account has been disabled in the database.
eMalformedRequest	Either username or domain is empty.
eTemporaryAuthFailure	Database cannot be connected. The connection is unexpectedly lost.
eSuccess	The password returned successfully.

The following code example shows how to use getUsersPass().

Example:

```
bool start(const std::string &auth, const std::string
&domain, SaslErrorCode &error, std::string &challenge)
{
...
SaslErrorCode sasl_error = eSuccess;
std::string db_pass("");
sasl_error = _context.getUsersPass(user, domain,
db_pass);
if (sasl_error != eSuccess)
{
        error = sasl_error;
        return false;
}

if (db_pass.compare(pass) != 0)
{
        error = eNotAuthorized;
        return false;
}
error = eSuccess;
_username = user;
return false;
```

```
}
```

getAuthzEnabled

Example:

```
bool getAuthzEnabled();
```

Some mechanisms might check whether VCS supports an authorization feature. Currently, VCS does not yet support the authorization identity (authzid). So, getAuthzEnabled() always returns false.

saslLog

Example:

```
void saslLog(LogLevel level, const char* channel,
const char* message);
```

This function writes logs into the SASL component log file.

Note: Only messages that have equal or higher levels than the SASL component log level will be written into the log.

Parameter	Note
level	There are six error log levels: SASL_ERROR, SASL_ALARM, SASL_WARNING, SASL_INFO, SASL_VERBOSE, and SASL_DEBUG.
channel	OPTIONAL. Helps to categorize the logger from the Sasl component log. If channel is NULL, the default channel format is "__FILE__:__LINE__", e.g.: "SaslComponent.cpp:162". The length of channel must be less than 512 bytes, including the terminating null byte ('\0').
message	The message body of logged text. The length of the message must be less than 1024 bytes, including the terminating null byte ('\0').

Mechanism Exceptions

The MechanismException function raises exceptions to the SASL component. The SASL component catches the exception to perform logging, deletes the mechanism instance, and returns error code 400 to the XMPP client.

Example:

```
if (bad happens)
{
throw MechanismException("Some exception
happened.");
return false;
}
```

SASL API References

This section provides reference information for the .hpp and .cpp files supported by the VCS SKD SASL API.

Mechanism.hpp

Mechanism

```
class Mechanism
{
public:
    Mechanism(SaslContext&);
    virtual ~Mechanism();

    virtual bool start(const std::string &auth,
const std::string &domain, SaslErrorCode &error,
std::string &challenge) = 0;
    virtual bool step(const std::string
&response, const std::string &domain,
SaslErrorCode &error, std::string &challenge) =
0;

    const std::string& getUserName() const;
    virtual const char* getMechanismName() const
= 0;
protected:
    SaslContext &_context;
    std::string username;
};
```

MechanismFactory

```
typedef Mechanism*
(*MechanismFactory)(SaslContext&);
```

MechanismInitFunc

```
typedef MechanismFactory
(*MechanismInitFunc)(SaslContext&, std::string&);
```

SASLErrorCode

```
enum SASL_AUTH_API SaslErrorCode {
    eSuccess = 0,
    eAborted,               //aborted
    eAccountDisabled,       //account-disabled
    eCredentialsExpired,    //credentials-expired
    eEncryptionRequired,    //encryption-required
    eIncorrectEncoding,     //incorrect-encoding
    eInvalidAuthzid,        //invalid-authzid
    eInvalidMechanism,      //invalid-mechanism
    eMalformedRequest,      //malformed-request
    eMechanismTooWeak,      //mechanism-too-weak
    eNotAuthorized,         //not-authorized
    eTemporaryAuthFailure,  //temporary-auth-
failure
};
```

Note: SASL Errors are defined at http://tools.ietf.org/html/rfc6120#section-6.5. The error code should be returned from the mechanism plug-in to indicate the success or failure authentication result.

MechanismException

```
class SASL_AUTH_API MechanismException : public
std::exception
{
public:
    MechanismException(const std::string &error)
throw()
            : _error(error)
    { }
    ~MechanismException() throw()
    { }

    const char* what() const throw()
    {
        return _error.c_str();
    }
private:
   // contains any errors reported by Mechanism
    std::string _error;
};
```

Note: MechanismException is a helper class to raise exception to SASL component. The SASL component catches the exception to perform logging and cleanup works.

Mechanism.cpp

Mechanism.cpp

```
#include "Mechanism.hpp"

using namespace sasl_auth;

Mechanism::Mechanism(SaslContext
&context)
    : _context(context)
{
}

Mechanism::~Mechanism()
{
}
```

SaslContext.hpp

SaslContext

```
enum LogLevel
{
SASL_ERROR,
SASL_ALARM,
SASL_WARN,
SASL_INFO,
SASL_VERBOSE,
SASL_DEBUG,
};

class SaslContext
{
public:
    virtual enum SaslErrorCode
getUsersPass(const std::string &user,

const std::string &domain,

std::string& pass) = 0;
virtual bool getAuthzEnabled() = 0;
virtual void saslLog(LogLevel level, const
char* channel, const char* message) = 0;
    virtual ~SaslContext();
protected:
    // SaslContext may be constructed by
derived classes only
    SaslContext();

private:
SaslContext(const SaslContext&);
    void operator=(const SaslContext&);
};
```

SaslContext.cpp

SaslContext.cpp

```
#include
"SaslContext.hpp"

using namespace
sasl_auth;

SaslContext::SaslConte
xt()
{
    // deliberately
empty
}

/* virtual */
SaslContext::~SaslCont
ext()
{
    // deliberately
empty
}
```

External HTTP SASL Plugin User Information

This section specifies how to install, configure, and debug the External HTTP SASL plugin on VCS.

External HTTP SASL Plugin Installation

The External HTTP SASL plugin is packaged as a COP file. The plugin installation should follow the standard COP file installation procedure, as described in *Installing COP Files for the Videoscape Control Suite* (part number OL-27753).

Prepare and Import the COP Template File

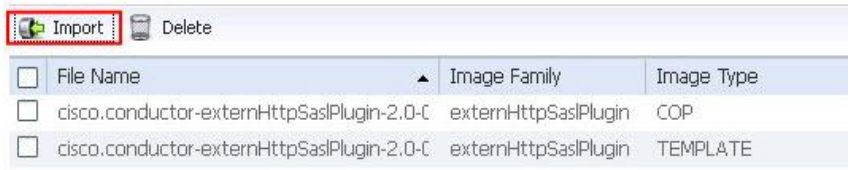
The External HTTP SASL plugin COP file needs a configuration template file which helps the CMC to create a configuration table. The configuration table requires the user to define the external HTTP server URL for user login authentication.

The configuration template file should be named **`cisco.conductor-externHttpSaslPlugin-{version}.tmp.xml`**.

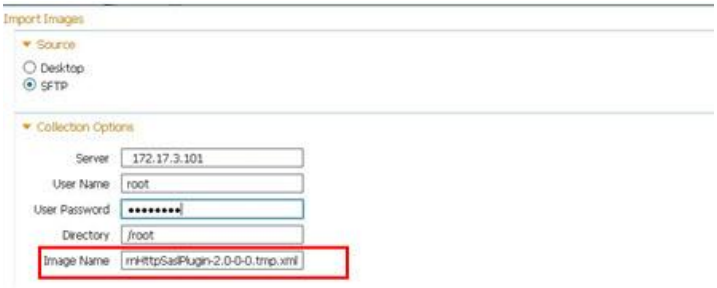
```
<!-- cisco.conductor-externHttpSaslPlugin-{version}.tmp.xml -->
<configuration>
 <node-attributes>
<column attribute="server-url"
     helptxt="Login Authentication HTTP Server URL " validation="text"
     label="HTTP Server URL"/>
 </node-attributes>
</configuration>
```

1. Log into the CMC.
2. Select **Operate --> Download**.

3 Click **Import**.



4 Import the cisco.conductor-externHttpSaslPlugin-{version}.tmp.xml file to the CMC through either the Desktop or the SFTP method.



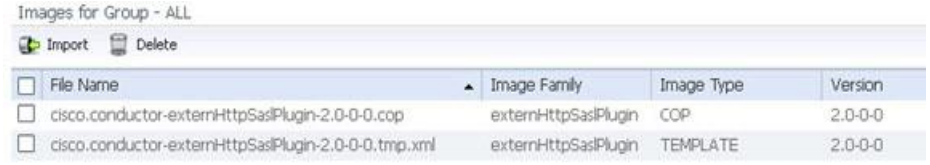
Install the SASL Plugin COP File

After the configuration template file has been imported, the COP file can be installed onto VCS.

1 Download the SASL plugin COP file and install it onto VCS by the standard COP file installation procedure described in *Installing COP Files for the Videoscape Control Suite* (part number OL-27753).

Note: The External SASL plugin COP file is usually named **`cisco.conductor-externHttpSaslPlugin-<Version>.cop`**.

2 Check the imported file list and make sure both the template file and the COP file are loaded on the CMC.



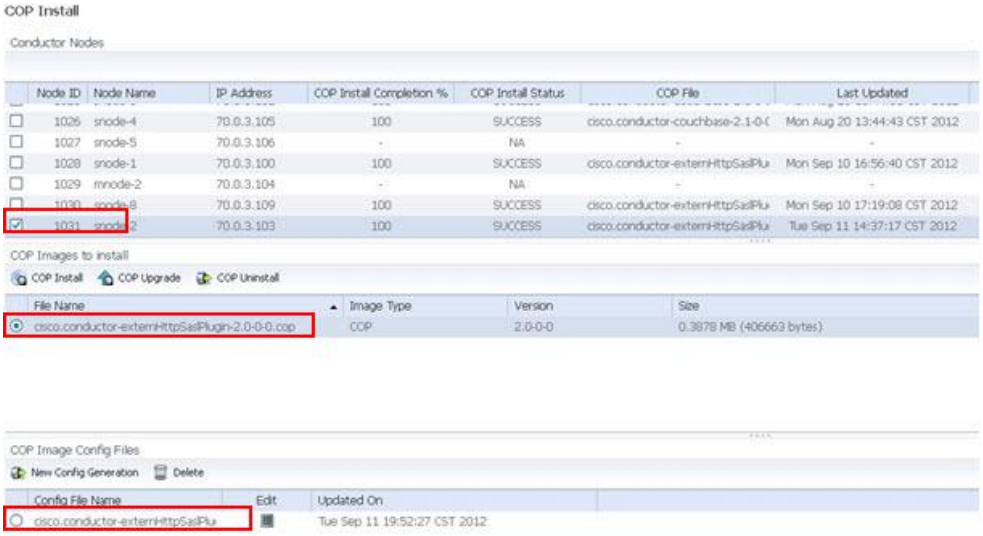
3 Select **COP Install**, under **Operate**.

4 In the **Conductor Nodes** table, select the node on which the SASL plugin will be installed.

5 In the **COP Images to install** table, select the **`cisco.conductor-externHttpSaslPlugin-{version}.cop`** file.

6 In the COP Image Config Files table, select the configuration file corresponding to the COP file node on which the SASL Plugin will be installed.

Note: If no configuration file exists, first click **New Config Generation** to generate a new configuration file.



Important:

- The configuration template file must be imported to the CMC before the COP file can be installed. Otherwise, the configuration file cannot be created.
- The configuration file is named **`cisco.conductor-externHttpSaslPlugin-{version}-{node id}.cfg.xml`**. The user must select the configuration file with the "node id" in its file name hat corresponds to the node on which the COP file will be installed.

7 Click **Edit**, beside the configuration file name, and and enter the URL of the external login authentication HTTP server.

Important: Be sure that the server URL starts with **http://** or **https://**. Otherwise, the COP file installation will fail.

Example: `http://<Magento server>/customerapi/account/login`

8 Click **COP Install**.

9 Check the Conductor Nodes table to make sure the COP file has successfully installed.

Deploy the SASL Plugin

In order to create a valid SASL plugin package, you need to follow these rules.

- The developed SASL plugin must be packaged as a tar file with "SASL" as the prefix of the filename, such as **`SASL<plugin name>.tar`**.
- The SASL plugin package should provide MD5 authentication for the SASL plugin library.
- The checksum file must be name ".checksum", and put it in the root directory of the SASL plugin package.
- Be sure to package the external HTTP SASL plugin as **`SASLexternhttp.tar`**.

```
[root@localhost saslplugin]# tar -tf SASLexternhttp.tar
externhttp-sasl-plugin.xml
externhttp.so
.checksum
```

The contents of the checksum file are illustrated below.

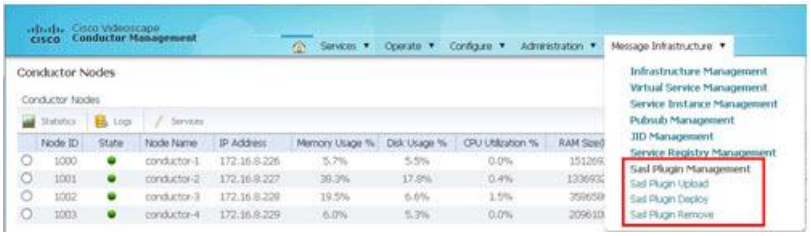
```
[root@localhost saslplugin]# less .checksum
e4750b0993fadee92ddfcddd1bcc8ac3  externhttp-sasl-plugin.xml
9e3a96864e247ce896d9a6c2455dfd97  externhttp.so
```

Notes:

- You can create the checksum file with the md5sum command.
  Example: `md5sum externhttp.so externhttp-sasl-plugin.xml > .checksum`
- Each SASL plugin package must have the valid .checksum file. The back end of the VCS system will authenticate the checksum file to make sure the SASL plugin is original.

Upload, Deploy, and Remove the SASL Plugin from the CMC

The following image shows the menu selections applicable to uploading, deploying, and removing the SASL plugin.



- **Upload** — You can upload the SASL plugin from the destop or via sftp. The SASL plugin must be packaged as a tar file with *SASL* as the prefix of the filename, such as **`SASL<plugin name>.tar`**.
- **Deploy** — The SASL plugin will be deployed to whatever service node you select. After the download, the SASL plugin will be extracted to this path: **`opt/cisco/conductor/saslplugins/<SASL plugin package name>`**.
  Example:

```
[root@sasl-siyao conductor]# ls -al
saslplugins/SASLexternhttp.tar/
total 1324
drwxr-xr-x  2 root   root    4096 Nov 19 20:04 .
drwxr-xr-x 14 root   root    4096 Nov 19 20:04 ..
-rw-r--r--  1 321735   25     109 Oct 21 22:04 .checksum
-rw-r--r--  1 321735   25      33 Oct 11 19:30 externhttp-
sasl-plugin.xml
-rwxr-xr-x  1 321735   25 1332153 Oct 16 04:43
externhttp.so
```

Configure the External SASL Plugin

Re-Config enables the operator to change the SASL setting from the CMC GUI. The operator can add or remove SASL Mechanisms by clicking the **Add** or **Remove** buttons.

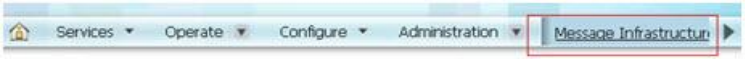
Msginfra Re-Config enables the operator to complete the all SASL setting steps on the CMC GUI. After deploying the SASL plugins, the operator can do these things:

- Enable or disable one or many plugin packages
- Set the Mechanism for these packages
- Set the library name from the CMC GUI
- Set the load from the GUI

Note: The operator can set up to 10 Mechanisms.

Before configuring SASL, the operator must be sure the packages have been deployed into client CM nodes. Otherwise, the configuration will not take effect and a TRAP will be triggered to the CMC GUI. SASL changes will only take effect after restarting CM. The configuration process does not restart CM automatically; the operator has to reboot the related router from the Node Management page in the CMC.

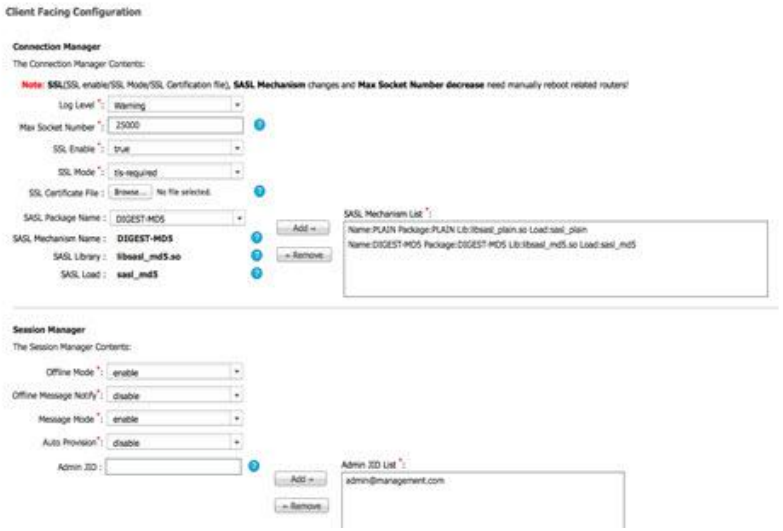
1. Log into the CMC.
2. Switch to Message Infrastructure view by clicking **Message Infrastructure** in the upper right corner.



   Note: SASL has two methods of configuration, Client Facing and Service Facing.
3. From **Message Infrastructure**, select either **Client Facing Configuration** or **Service Facing Configuration**.

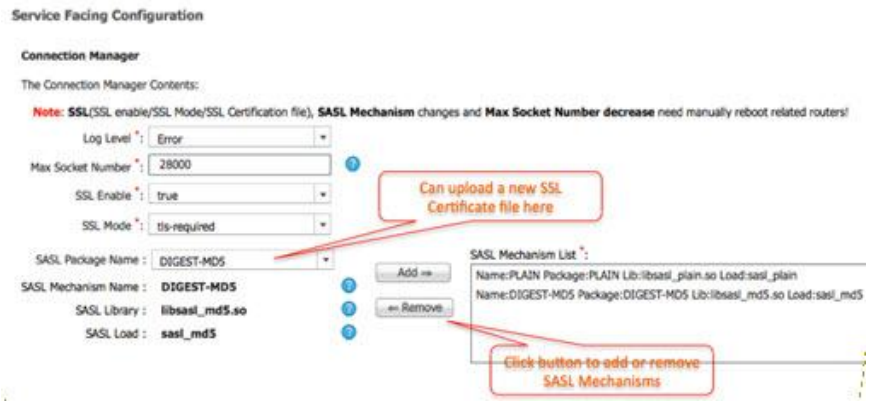
Client Facing Configuration

Select **Client Facing Configuration**.



Connection Manager

The Connection Manager contains Service CM component-related configurations, such as SSL and SASL, as shown in the following illustration.



Configure these parameters:

- **Log Level —** Re-Config supports the change of the log level setting on all Service CMs. The new setting will take effect without the restart of CM. The available log level values are: **debug**, **verbose**, **info**, **warning**, **alarm**, and **error**.
- **Max Socket Number —** The number of sockets that every service CM can maintain. The default value is 25000. The available range is [1,100000].

  Note: If the number is smaller than the current configuration, the operator needs to restart the router for the setting to take effect. Otherwise, the value will take effect without restarting the service CM component.
- **SSL Enable —** Enables/disables SSL of the Connection Manager, if **SSL Enabled** is **false.** VCS services that do not support TLS can still connect to the server.

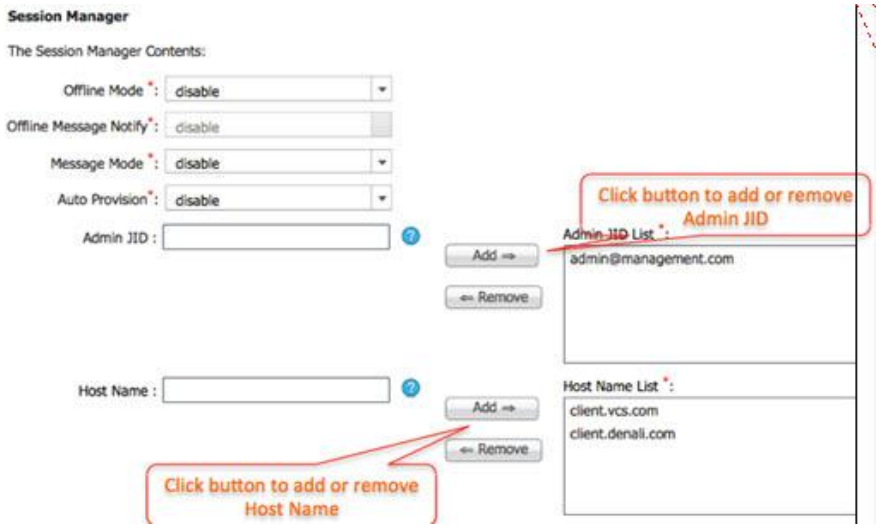
- **SSL Mode —** Only available when **SSL Enable** is **true**. There are two options:
  - **tls-required —** Enables TLS (transport layer security). VCS services that support TLS can connect to the server securely. VCS services that do not support TLS can still connect to the server. This mode does not require a secure connection.
  - **tls —** The same as the tls option, except that the client must support TLS. VCS services that do not support TLS cannot connect to the server.
- **SSL Certificate File —** Re-Config supports changes to the SSL certificate file of the VCS. The operator can select a new certificate file by clicking **browse**. If it is set, the new certificate file will be uploaded to /opt/cisco/conductor/xcp/certs/tls in the VCS node. The Connection Manager's certificate fie path is also set to /opt/cisco/conductor/xcp/certs/tls.

When adding a new SASL Mechanism, the following items should be configured:

- **SASL Package Name —** The page will list all available packages supported by VCS. The operator can select a package, set the Mechanism name, library, and load of this package. Then, the operator should click **Add** to add the mechanism. The package should have been deployed to VCS from the External SASL Plugin pages. The menu path is **Message Infrastructure --> External SASL Plugin --> Deploy**.
- **SASL Mechanism Name —** The name for new SASL Mechanism to be added. It can include digit, uppercase letter,"_" and "-". The length can be from 1 to 20.
- **SSL Library —** The *.so* file name of the selected package. It must start with a letter or digit, and end with ".so". It can contain digits, letters,"_", and "-". Total length can be from 1 to 32.
- **SSL Load —** The main function of the *.so* file. It must start with a letter or "_". It can contain digits, letters, and "_". The total length can be from 1 to 32.

Session Manager

The Session Manager contains Client JSM and its SDNS-related configurations. All configurations take effect without a restart of the router.



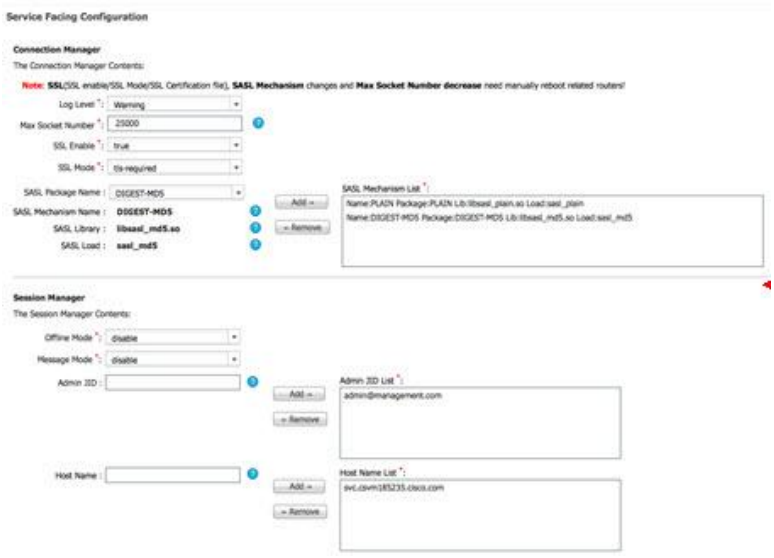
- **Offline Mode** — If Enabled, the client can receive and retrieve offline messages. Otherwise, the client loses offline message functions.
- **Offline Message Notify** — Only available when **Offline Mode** is enabled. If enabled, VCS delivers offline messages of the Pubsub to Unified Notification Gateway (UNG) service.
- **Message Mode** — If enabled, all clients with the same JID, but with different resources, will receive messages. Otherwise, only the client that has the highest priority can receive messages.
- **Auto Provision** — Select Enable if you want users who authenticate via SASL or x509 to be created automatically in the JSM database when they create a new session.
- **Admin JID** — The Admin JID number has a range of [1-100]. The Client JSM must have at least one Admin JID at least and no more than 100.
- **Host Name** — This is the client session domain name. The operator can set up to 100 host names.

  Note: If the client-facing hostname is changed, all client JIDs with previous the host name will be invalid and can no longer connect to the VCS.

Service Facing Configuration

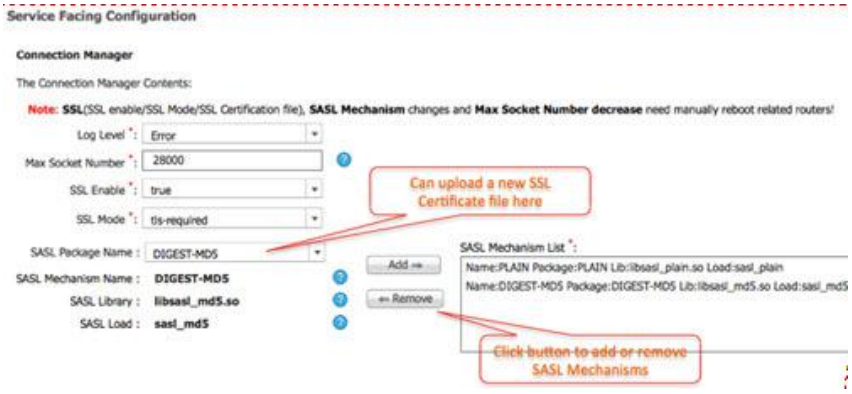
Select **Service Facing Configuration**.

The service facing configurations refers to all configuration items related the VCS service login, message sending, and message receiving. The related components/plugins that are changed are: CM, JSM, and SDNS.



Connection Manager

The Connection Manager contains service CM component-related configurations, such as SSL and SASL.



Configure the following parameters:

- **Log Level** — Re-Config supports the change of the log level setting on all Service CMs. The new setting will take effect without the restart of CM. The available log level values are: **debug**, **verbose**, **info**, **warning**, **alarm**, and **error**.

- **Max Socket Number —** The number of sockets that every service CM can maintain. The default value is 25000. The available range is [1,100000].

  Note: If the number is smaller than the current configuration, the operator needs to restart the router for the setting to take effect. Otherwise, the value will take effect without restarting the service CM component.

- **SSL Enable —** Enables/disables SSL of the Connection Manager, if **SSL Enabled** is **false.** VCS services that do not support TLS can still connect to the server.

- **SSL Mode —** Only available when **SSL Enable** is **true**. There are two options:

  - **tls-required —** Enables TLS (transport layer security). VCS services that support TLS can connect to the server securely. VCS services that do not support TLS can still connect to the server. This mode does not require a secure connection.

  - **tls —** The same as the tls option, except that the client must support TLS. VCS services that do not support TLS cannot connect to the server.

- **SSL Certificate File —** Re-Config supports changes to the SSL certificate file of the VCS. The operator can select a new certificate file by clicking **browse**. If it is set, the new certificate file will be uploaded to /opt/cisco/conductor/xcp/certs/tls in the VCS node. The Connection Manager's certificate fie path is also set to /opt/cisco/conductor/xcp/certs/tls.

When adding a new SASL Mechanism, the following items need to be configured:

- **SASL Package Name —** The page will list all available packages supported in VCS. The operator can select a package, set the mechanism name, the library, and the load of this package. Then, click **Add** to add the mechanism. The package should have been deployed to VCS from the External SASL Plugin pages. The menu path is **Message Infrastructure --> External SASL Plugin --> Deploy**.

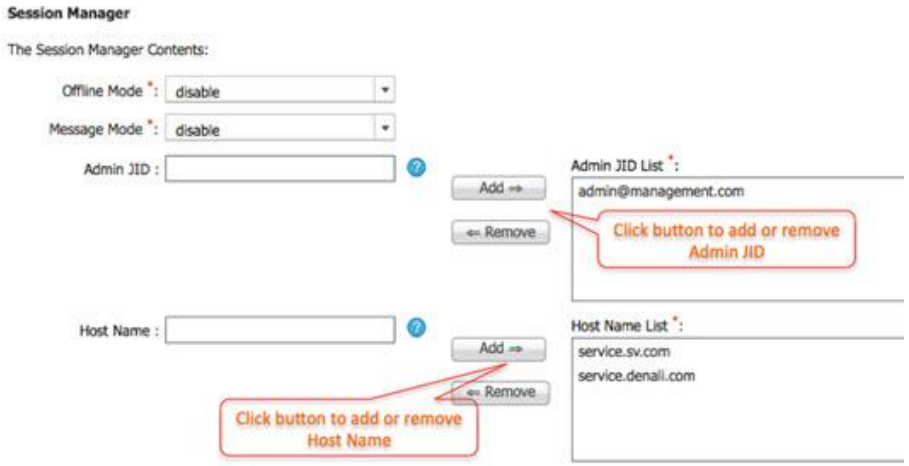
- **SASL Mechanism Name —** The name for new SASL Mechanism to be added. It can include digit, uppercase letter,"_" and "-". The length can be from 1 to 20.

- **SSL Library —** The *.so* file name of the selected package. It must start with a letter or digit, and end with ".so". It can contain digits, letters,"_", and "-". Total length can be from 1 to 32.

- **SSL Load —** The main function of the *.so* file. It must start with a letter or "_". It can contain digits, letters, and "_". The total length can be from 1 to 32.

Session Manager

The Session Manager contains Service JSM and SDNS-related configurations. All configurations can take effect without a restart of the router.



- **Offline Mode —** If Enabled, the client can receive and retrieve offline messages. Otherwise, the client loses offline message functions.
- **Message Mode —** If enabled, all clients with the same JID, but with different resources, will receive messages. Otherwise, only the client that has the highest priority can receive messages.
- **Admin JID —** The Admin JID number has a range of [1-100]. The Client JSM must have at least one Admin JID at least and no more than 100.
- **Host Name —** This is the client session domain name. The operator can set up to 100 host names.

  Note: If the client-facing hostname is changed, all client JIDs with previous the host name will be invalid and can no longer connect to the VCS.

HTTP Server Interface

The message interface between the External SASL plugin and HTTP authentication server is defined below.

- Authentication Request example:
  < Authentication Server URL>?username=test2012@cisco.com&password=test123
- Response example:
  {"status":true,"message":"1ph0aa36emjj8t1vtkkbfdfd30"}

The user defines the URL, which the SASL plugin reads from the configuration file. The configuration file is located at **/opt/cisco/conductor/etc/externhttp-sasl-plugin.xml**.

Example using the Magento HTTP server:

```
<!-- externhttp-sasl-plugin.xml -->

<?xml version='1.0' encoding='UTF-8'?>

<server-url> http://<Magento
server>/customerapi/account/login</server-url>
```

The software developer should be aware of the following:

- The HTTP post data format is the same as illustrated in the Authentication Request example, above:
  username='username' and password='password'
- The response is a JSON string.
  {"status":true,"message":"1ph0aa36emjj8t1vtkkbfdfd30"}

The value of the message is the Session ID. The message is not a requirement of the SASL plugin. The SASL plugin checks only the value of the status to determine authentication success or failure.

Security Considerations

- The External HTTP SASL plugin needs to POST the username and password to the external HTTP server for authentication. The client application should follow the PLAIN mechanism that encodes the user password by base64. The External HTTP SASL plugin will decode the password into clear text and post it to the external HTTP server. The posted data looks similar to **`username=1test&password=test`**.

  To enhance the security of the password transmission, the client application can encrypt the password first and encode by base64, and then pass it to the External HTTP SASL plugin. The HTTP server should also decrypt the password after receiving the authentication request from the SASL plugin.
- To enhance the security, VCS requires that the client implement TLS protocol to *handshake* with VCS.

For Information

If You Have Questions

If you have technical questions, contact Cisco Services for assistance. Follow the menu options to speak with a service engineer.

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA

http://www.cisco.com
Tel: 408 526-4000
800 553-6387
Fax: 408 527-0883

November 2013 Part Number OL-26999-01