Genesys Interaction Recording Solution Guide

Configuring Security

4/3/2026

Contents

Configuring Security

Web Services adheres to the standards described in the Open Web Application Security Project (OWASP) Top 10—see the OWASP website for details—and has adopted several methods of ensuring security, for example:

- Errors are logged locally to prevent information leakage through API requests.
- User sessions have a timeout option.
- Cross Site Request Forgery Protection

Read on for details about the additional security configurations that Interaction Recording Web Services includes.

Transport Layer Security (TLS)

Complete the procedures below to configure TLS for connections received by Interaction Recording Web Services, and for connections from Interaction Recording Web Services to the following:

- Configuration Server
- SIP Server
- Interaction Server
- WebDAV
- Recording Crypto Server
- SpeechMiner Interaction Receiver
- Cassandra

Configuring TLS on the Server Side for Interaction Recording Web Services

1. Enable SSL on Jetty by configuring the **SSL** section of the **application.yaml** file using the following parameters:

   ```
   enableSsl: true
   ssl:
      port: 443
      keyStorePath: keystore
      keyStorePassword: storepwd
   ```

   For more information on the parameters, see ssl.

> **Important**
>
> On Unix-based systems, port 443 is protected; typically, only the superuser root can open it. For security reasons, it is not recommended to run the server as root. Therefore, Genesys recommends that you bind it to a non-protected port. Typically, any port above 1024 can be used (for example, you could set it to 9443). If you want to continue using port 443, see Setting Port 80 Access for a Non-Root User in the Jetty documentation.

2. Acquire the certificate and private keys.
3. To load a certificate and private keys (`jetty.crt`), navigate to the `GWS_HOME/etc` directory and run the following commands:

   ```
   keytool -keystore keystore -import -alias jetty -file jetty.crt -trustcacerts
   ```

4. When prompted for the keystore password, enter the default: `storepwd`
5. Restart Interaction Recording Web Services (Web Services).

To create a self-signed certificate for non-production purposes:

1. Run the following in `GWS_HOME/etc`:

   ```
   keytool -genkey -keyalg RSA -keystore keystore -alias jetty -ext
   SAN=dns:<server_dns_name>,ip:<server_ip_address>
   ```

2. When prompted for the keystore password, enter the default: `storepwd`

   For more information about configuring SSL, see Configuring SSL/TLS.

To change the certificate:

1. Remove the existing certificate using the following command:

   ```
   keytool -keystore keystore -delete -alias jetty
   ```

2. Acquire the certificate and private key in a X509 PEM file (for example, `jetty.crt`).
3. Load the certificate using the following command:

   ```
   keytool -keystore keystore -import -alias jetty -file jetty.crt -trustcacerts
   ```

4. Restart Interaction Recording Web Services (Web Services).

To change the keystore password:

1. Execute the following command:

   ```
   keytool -keystore keystore -storepasswd
   ```

2. Encode the new password using the following command:

   ```
   java -cp lib/jetty-http-xxx.jar:lib/jetty-util-xxx.jar
   org.mortbay.jetty.security.Password <your password here>
   ```

Configuring TLS connections to Configuration Server, SIP Server, and Interaction

Server

Interaction Recording Web Services can use a secured Transport Layer Security (TLS) connection mechanism to connect to Configuration Server, Interaction Server, and SIP Server. When configured, Interaction Recording Web Services connects to secure ports on Configuration Server, Interaction Server, and SIP Server; verifies the server's certificate; and encrypts/decrypts network traffic. You can configure secured connections to Configuration Server, Interaction Server, and SIP Server in the following ways:

- Minimal configuration for Configuration Server, SIP Server, and Interaction Server
- Validate the Certificate Against the CA

Note that each connection is configured independently, but a similar mechanism is used to configure each connection.

Prerequisites

Before configuring Interaction Recording Web Services, make sure the secure port on the server is configured as described in Introduction to Genesys Transport Layer Security in the Genesys Security Deployment Guide and that certificates for the server and the Certificate Authority are configured and available.

Minimal configuration for Configuration Server, SIP Server, and Interaction Server

In this configuration, Interaction Recording Web Services does not check the certificate against the Certificate Authority, but all traffic is encrypted. To configure Interaction Recording Web Services with minimal configuration, all you need to do is configure a connection to a secured port on Configuration Server, SIP Server, and Interaction Server. You can do this using *either* of the following methods:

- For the initial connection to Configuration Server, set the tlsEnabled option to `true` in the **onPremiseSettings** section of the RWS **application.yaml** file on the RWS node that is configured to be the sync node. This creates a secured connection to Configuration Server the first time Interaction Recording Web Services starts.
- For an environment that is already configured with Configuration Server synchronization enabled, you can make changes with Configuration Server as described in the Genesys Security Deployment Guide. These changes are synchronized back to the Cassandra database from Configuration Server.

> **Important**
>
> Configuration Server supports the auto-upgrade port connection for secure communication from other GIR components; however, a secure port (listening mode of type secured) must be used for connectivity from RWS to the Configuration Server.

Ensure that connections from the Cluster Application being used by RWS (either IRWS_Cluster or WS_Cluster; see Installing Interaction Recording Web Services for more information) specify the appropriate secure port on each of the servers.

Validate the Certificate Against the CA

The procedure to validate the certificate against the CA is common to Configuration Server, SIP Server and Interaction Server.

Ensure you have completed the procedure described in the Minimal configuration for Configuration Server, SIP Server, and Interaction Server section.

To support the client-side certificate check, Interaction Recording Web Services needs the public key for the Certificate Authority (CA). Interaction Recording Web Services supports the PEM and JKS key storage formats, but recommends using JKS because it's compatible with both Cassandra and HTTPS.

To validate the certificate against the CA, specify the path to a file containing the trusted CA in the **caCertificate** parameter in the **application.yaml** file. By specifying this parameter, this CA will be checked against the server CA for validation.

> **Important**
>
> Only a single CA can be used to validate the certificate from Configuration Server, SIP Server, and Interaction Server.
>
> If the configured server certificate matches the hostname of the server for any of the following fields, then Interaction Recording Web Services will validate the certificate.
>
> - Issuer CN
> - Subject CN
> - Subject Alternative Name DNS

To validate the certificate against the CA, complete the following steps.

> **Important**
>
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow your own company's security policies for creating and signing certificates.

Start

1. If you plan to use a JKS file, you can generate it from a PEM file by importing the PEM certificate, as shown here:

   ```
   keytool -importcert -file ca_cert.pem -keystore ca_cert.jks
   ```

2. Once you have the **ca_cert.jks** file, place it in a location accessible from your Interaction Recording Web Services host, such as:
   - A local folder on the Interaction Recording Web Services host

- A shared folder

3. Configure the following options in the **serverSettings** section of the **application.yaml** file:
   - For a PEM file, set caCertificate to the location of the file. For example:

     ```
     caCertificate: /opt/ca_cert.pem
     ```

   - For a JKS file, set caCertificate to the location of the file and set jksPassword to the password for the key storage. For example:

     ```
     caCertificate: /opt/ca_cert.jks
     jksPassword: pa$$word
     ```

End

Configuring TLS for Connections to WebDAV

By default, Interaction Recording Web Services checks the WebDAV server's certificate against a Certificate Authority using the Java default trustStore **caCerts**. To configure Interaction Recording Web Services with a customized trustStore configuration or to disable certificate validation, set the following options in the **application.yaml** configuration file.

Name	Parent	Value	Default
webDAVTrustedCA	serverSettings	Configures TLS certificate validation when Interaction Recording Web Services connects to a WebDAV server. Valid values are `true`, `false`, or a path to a file containing one or more CA certificates. • If set to `true`, the certificate that WebDAV presents will be validated by **caCerts** in `$JAVA_HOME/jre/lib/security`. • If set to `false`, the certificate that WebDAV presents will not be validated. • Any other value is considered as a path to a file containing a certificate for a Certificate Authority and RWS will use it to validate the WebDAV certificate. Both PEM and JKS	`true`

Name	Parent	Value	Default
		key storage formats are supported. If the specified file does not exist, Interaction Recording Web Services will exit during initialization.	
webDAVJksPassword	serverSettings	The password for the key storage for WebDAV if the specified CA file is in JKS format. You can specify an encrypted password. For more information on encrypting a password, see Password Encryption.	Empty

Configuring TLS for Connections to Recording Crypto Server

By default, Interaction Recording Web Services checks the Recording Crypto Server's certificate against a Certificate Authority using the Java default trustStore **caCerts**. To configure Interaction Recording Web Services with a customized trustStore configuration or to disable certificate validation, set the following options in the **application.yaml** configuration file.

Name	Parent	Value	Default
rcsTrustedCA	serverSettings	Configures TLS certificate validation when Interaction Recording Web Services connects to the Recording Crypto Server. This property can be set to `true`, `false`, or a path to a file containing one or more CA certificates. • If set to `true`, the certificate that RCS presents will be validated by **caCerts** in `$JAVA_HOME/jre/lib/security`. • If set to `false`, the certificate that RCS presents will not be validated.	`true`

Name	Parent	Value	Default
		• Any other value is considered as a path to a file containing one or more CA certificates and RWS will use it to validate the RCS certificate. The key storage format can be either PEM or JKS. If the specified file does not exist, Interaction Recording Web Services will exit during initialization.	
rcsJksPassword	serverSettings	The password for the key storage for RCS if the specified CA file is in JKS format. You can specify an encrypted password. For more information on encrypting a password, see Password Encryption.	Empty

Configuring TLS for Connections to SpeechMiner Interaction Receiver

By default, Interaction Recording Web Services checks the SpeechMiner Interaction Receiver's certificate against a Certificate Authority using the Java default trustStore **caCerts**. To configure Interaction Recording Web Services with a customized trustStore configuration or to disable certificate validation, set the following options in the **application.yaml** configuration file.

Name	Parent	Value	Default
speechMinerTrustedCA	serverSettings	Configures TLS certificate validation when Interaction Recording Web Services connects to SpeechMiner Interaction Receiver. Valid values are `true`, `false`, or a path to a file containing one or more CA certificates. • If set to `true`, the certificate that the SpeechMiner Interaction Receiver	`true`

Name	Parent	Value	Default
		presents will be validated by **caCerts** in `$JAVA_HOME/jre/lib/security`. • If set to `false`, the certificate that the SpeechMiner Interaction Receiver presents will not be validated. • Any other value will be considered as a path to a file containing one or more CA certificates and RWS will use it to validate the SpeechMiner Interaction Receiver certificate. The key storage format can be either PEM or JKS. If the specified file does not exist, Interaction Recording Web Services will exit during initialization.	
speechMinerJksPassword	serverSettings	The password for the key storage for SpeechMiner Interaction Receiver if the specified CA file is in JKS format. You can specify an encrypted password. For more information on encrypting a password, see Password Encryption.	Empty

Configuring TLS for Connections with Cassandra

Genesys supports Transport Layer Security (TLS) for connections from Interaction Recording Web Services to Cassandra and between Cassandra nodes. You can configure secured connections for the following scenarios:

- Secure Connections from Interaction Recording Web Services to Cassandra
- Secure Connections from Interaction Recording Web Services to Cassandra 4

- Secure Connections between Cassandra Nodes

Secure Connections from Interaction Recording Web Services to Cassandra

Prerequisites

- You have installed Bash, Java keytool, and OpenSSL.

Complete the following steps to configure TLS for connections from Interaction Recording Web Services to Cassandra.

> **Important**
>
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow your own company's security policies for creating and signing certificates.

Start

1. Create the server-side keystore with a self-signed certificate and the client-side truststore — which contains the public part of server certificate — with the following commands:

```
#!/bin/bash
#generate keypair
keytool -genkeypair -alias cassandra -keyalg RSA -keysize 1024 -dname
"CN=<Cassandra node hostname>, OU=Test, O=Test Ltd, C=US" -keystore
server.jks
-storepass password -keypass password
#export certificate
keytool -exportcert -alias cassandra -file client.pem -keystore
server.jks -storepass password -rfc
#create client truststore and import certificate
keytool -importcert -alias cassandra -file client.pem -keystore
client.jks -storepass password -noprompt
```

2. Create a self-signed root authority, use it to sign the server certificate, store it to **server.jks** and create the client-side truststore, which trusts all certificates signed with root authority. Run the following commands:

```
#!/bin/sh

#generate self-signed root certificate
keytool -genkeypair -alias root -keyalg RSA -keysize 1024 -validity 3650 -dname
"CN=TestRoot, OU=Dev, O=Company, C=US" -keystore root.jks
-storepass password -keypass password

#export root certificate
keytool -exportcert -alias root -file root.crt -keystore root.jks -storepass password

#generate server-side certificate
keytool -genkeypair -alias server -keyalg RSA -keysize 1024 -validity 3650 -dname
"CN=TestServer, OU=Dev, O=Company, C=US"
-keystore server.jks -storepass password -keypass password
```

```
#create the sign request for server certificate
keytool -certreq -alias server -keystore server.jks -file server.csr -storepass password
-keypass password

#export private key of root auth: need later for signing the server certificate
keytool -v -importkeystore -srckeystore root.jks -srcalias root -destkeystore root.p12
-deststoretype PKCS12 -noprompt
-destkeypass password -srckeypass password -destalias root -srcstorepass password
-deststorepass password

openssl pkcs12 -in root.p12 -out private.pem -password pass:password -passin
pass:password -passout pass:password
rm root.p12

#sign the certificate
openssl  x509  -req  -CA   private.pem  -in server.csr -out server.crt  -days 3650
-CAcreateserial -passin pass:password
rm private.pem
rm private.srl
rm server.csr

#import root certificate to client side trust store
keytool -importcert -alias root -file root.crt -keystore client.jks -storepass password
-noprompt

#import root certificate to server side key store
keytool -importcert -alias root -file root.crt -keystore server.jks -storepass password
-noprompt
rm root.crt

#import certificate sign reply into server-side keystore
keytool -import -trustcacerts -alias server -file server.crt -keystore server.jks
-storepass password -keypass password
rm server.crt
```

3. Configure Cassandra to use your generated certificates for the client connection by setting the `client_encryption_options` in the **cassandra.yaml** file. For example:

```
client_encryption_options:
    enabled: true
    keystore: <absolute path to server.jks file>
    keystore_password: password
    #the password specified in while creating storage
    # For the purpose of the demo the default settings were used.
    # More advanced defaults below:
    #protocol: TLS
    #algorithm: SunX509
    #store_type: JKS
    #cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA]
```

> **Important**
>
> To enable support for encryption, you must have the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction installed.

4. Confirm the Cassandra nodes can start successfully:
   a. Edit the **conf/log4j-server.properties** file and uncomment the following line:

```
log4j.logger.org.apache.cassandra=DEBUG
```

b. Start Cassandra and check the logs. If the configuration was successful, you shouldn't see any errors.

c. Edit the **conf/log4j-server.properties** file and comment the following line to disable the functionality:

```
log4j.logger.org.apache.cassandra=DEBUG
```

d. Check that SSL-to-client is working successfully using cqlsh.

- Confirm that unsecured connections aren't possible by starting cqlsh locally—this forces it to connect to the Cassandra instance running on localhost. You should expect to see the exception in the cqlsh output.

```
cqlsh `hostname`
```

- Confirm that secured connections are possible by configuring cqlsh with SSL encryption. Create a PEM key which will be used in the **.cqlshrc** file:

```
// Create PEM for client
keytool -importkeystore -alias cassandra -srckeystore
server.jks -destkeystore server.p12 -deststoretype PKCS12
openssl pkcs12 -in server.p12 -out client.pem -nodes
```

- Create a **.cassandra/cqlshrc** file in your home or client program directory. The following settings must be added to the file as described below. When validate is enabled, the host in the certificate is compared to the host of the machine that it is connected to verify that the certificate is trusted.

```
[connection]
hostname = <hostname of Cassandra node>
port = 9042
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
certfile = /path/to/client.pem
# Optional, true by default
validate = true
```

- Verify that you can connect to Cassandra using cqlsh:

```
$ cqlsh --ssl
Connected to HTCC Cassandra Cluster at
ci-vm378.us.int.genesyslab.com:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native
protocol v4]
Use HELP for help.
```

- Configure Interaction Recording Web Services to use SSL with Cassandra. On each Interaction Recording Web Services node, edit the **application.yaml** file as follows:

```
cassandraCluster:
  useSSL: true
  trustStore: /path/to/client.jks
  truststorePassword: password
```

- Restart each Interaction Recording Web Services node:

```
systemctl restart gir
```

End

Secure Connections from Interaction Recording Web Services to Cassandra 4

Start

1. Create a self-signed root authority, use it to sign the server certificate, store it to server-side keystore and create the client-side truststore, which trusts all certificates signed with root authority. Run the following commands:

```
#!/bin/bash
#generate self-signed root certificate
keytool -genkeypair -alias root -keyalg RSA -keysize 2048 -validity 3650 -dname
"CN=TestRoot, OU=Dev, O=Company, C=US" -keystore root.jks -storepass password
-keypass password -ext KeyUsage=digitalSignature,keyCertSign -ext
BasicConstraints=ca:true

#export root certificate
keytool -exportcert -alias root -file root.crt -keystore root.jks -storepass password

#export private key of root auth: need later for signing the server certificate
keytool -v -importkeystore -srckeystore root.jks -srcalias root -destkeystore
root.p12 -deststoretype PKCS12 -noprompt -destkeypass password -srckeypass password
-destalias root -srcstorepass password -deststorepass password
openssl pkcs12 -in root.p12 -out private.pem -password pass:password -passin
pass:password -passout pass:password -legacy
rm root.p12

#import root certificate to client side trust store
keytool -importcert -alias root -file root.crt -keystore client.jks -storepass
password -noprompt

#import root certificate to server side key store
keytool -importcert -alias root -file root.crt -keystore server.jks -storepass
password -noprompt
rm root.crt
```

2. Create the server-side keystore with a self-signed certificate and the client-side truststore — which contains the public part of server certificate — with the following commands:

> **Important**
>
> In case of Multi Node Cassandra, the following steps should be executed for each Cassandra node.

```
#!/bin/bash
#generate server-side certificate
keytool -genkeypair -alias <node> -keyalg RSA -keysize 2048 -dname "CN=<Cassandra
node>, OU=Test, O=Test Ltd, C=US" -keystore server.jks -storepass password -keypass
password

#export certificate
keytool -exportcert -alias <node> -file <node>_client.pem -keystore server.jks
-storepass password -rfc

#create and import certificate to client truststore
keytool -importcert -alias <node> -file <node>_client.pem -keystore client.jks
-storepass password -noprompt

#create the sign request for server certificate
```

```
keytool -certreq -alias <node> -keystore server.jks -file <node>.csr -storepass
password -keypass password

#sign the certificate
openssl x509 -req -CA private.pem -in <node>.csr -out <node>.crt -days 3650
-CAcreateserial -passin pass:password

rm private.pem
rm private.srl
rm <node>.csr

#import certificate sign reply into server-side keystore
keytool -import -trustcacerts -alias <node_crt> -file <node>.crt -keystore server.jks
-storepass password -keypass password
rm <node>.crt
```

3. Continue from Step.3 in Secure Connections from Interaction Recording Web Services to Cassandra.

End

Secure Connections between Cassandra Nodes

When you enable SSL for connections between Cassandra nodes, you ensure that communication between nodes in the Cassandra cluster is encrypted, and that only other authorized Cassandra nodes can join the cluster.

The steps below show you how to create a single certificate to be used by all Cassandra nodes in the cluster. This simplifies cluster management because you don't need to generate a new certificate each time you add a new node to the cluster, which means you don't need to restart all nodes to load the new certificate.

> ### Important
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow your own company's security policies for creating and signing certificates.

Start

1. Generate a keystore and truststore. See Step 1 of Secure Connections from Interaction Recording Web Services to Cassandra for details.
2. On each Cassandra node in the cluster, set **server_encryption_options** in the **cassandra.yaml** file. For example:

```
server_encryption_options:
    internode_encryption: all
    keystore: <absolute path to keystore >
    keystore_password: <keystore password - somePassword in our sample>
    truststore: <absolute path to trustore>
    truststore_password: <truststore password - somePassword in our sample>
```

3. Check the Cassandra logs. If the configuration was successful, you shouldn't see any errors.

End

Configuring TLS for Connections with Elasticsearch and Elasticsearch basic authentication

Genesys supports Transport Layer Security (TLS) and authentication for connections from Interaction Recording Web Services (RWS) to Elasticsearch 8 and also for communication between Elasticsearch nodes. You can configure these connections manually or use Elasticsearch's built-in autoconfiguration tools.

- Secure Connections from Interaction Recording Web Services to Elasticsearch 8
- Secure Connections between Elasticsearch Nodes

Secure Connections from Interaction Recording Web Services to Elasticsearch 8

Configuring RWS with Elasticsearch Default certificates:

> **Important**
>
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow your own company's security policies for creating and signing certificates.

1. Elasticsearch 8 comes with security features enabled by default, even without any additional configuration. On the first startup, it automatically:
    1. Enables basic authentication.
    2. Generates a random password for the built-in elastic superuser.
    3. Enables HTTPS on localhost with a self-signed certificate.
    4. Generates default certificates and keystores for HTTP and transport layers.

    Once Elasticsearch 8 has successfully started, the following files are generated in the Elasticsearch configuration directory (/etc/elasticsearch or where the **elasticsearch.yml** resides).

File	Purpose
http_ca.crt	The Certificate Authority (CA) certificate that signed the HTTP layer certificate of this cluster. Clients connecting over HTTPS should trust this CA.
http.p12	A PKCS#12 keystore containing the key and certificate for the HTTP layer of this node.
transport.p12	A PKCS#12 keystore containing the key and certificate for the transport layer, used for secure communication between all nodes in the cluster.

2. RWS configuration sample
    1. We can use `http_ca.crt` to create a truststore for RWS connecting to Elasticsearch over HTTPS.
    2. We can find `<elastic` user password in Elasticsearch logs.

3. Configure Interaction Recording Web Services to use SSL with Elasticsearch. On each Interaction Recording Web Services node, edit the **application.yaml** file as follows:

```
elasticSearchSettings:
  ...
  restClient:
     nodes:
        - {host: <elastic-search-node1>, port: 9200}
      authentication:
        type: basic
        userName: elastic
        password: <ELASTIC_USER_PASSWORD>
      useSSL: true
      truststore: rws-client.jks # contains http_ca.crt
      truststorePassword: password
```

3. Restart each Interaction Recording Web Services node:

```
systemctl restart gir
```

Configuring RWS with Elasticsearch Manually generated certificates:

1. Generate Elasticsearch Node Keystore & Certificate

```
keytool -genkeypair -alias <elasticsearch-node> -keyalg RSA -keysize 2048 -dname
"CN=<Elasticsearch node hostname>, OU=Test, O=Test Ltd, C=US" -keystore es-server.jks
-storepass password -keypass password
keytool -exportcert -alias <elasticsearch-node> -file es-node.pem -keystore es-
server.jks -storepass password -rfc
```

2. Create Client-side Truststore for RWS

```
keytool -importcert -alias elasticsearch-node -file es-node.pem -keystore rws-
client.jks -storepass password -noprompt
```

3. To Create Additional Users in elasticsearch

```
<Elasticsearch_installation_directory>/bin/elasticsearch-users useradd <rws-user> -p
<password> -r <superuser>
```

4. Configure SSL options for Elasticsearch by editing the `elasticsearch.yml` in all elasticsearch nodes. For example:

```
xpack.security.enabled: true
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.keystore.path: /path/to/es-server.jks
xpack.security.http.ssl.keystore.password: password
```

5. Restart Elasticsearch services and check the logs. If the configuration was successful, you shouldn't see any errors.
    1. Verify cluster health:

```
 curl -u <rws-user>:<password> --cacert /path/to/ca.crt
https://<host>:9200/_cluster/health
```

6. Configure Interaction Recording Web Services to use SSL with Elasticsearch. On each Interaction Recording Web Services node, edit the **application.yaml** file as follows:

```
elasticSearchSettings:
  ...
  restClient:
     nodes:
```

```
    - {host: <elastic-search-node1>, port: 9200}
  authentication:
    type: basic
    userName: <ELASTIC_SEARCH_USER_NAME>
    password: <ELASTIC_SEARCH_PASSWORD>
  useSSL: true
  truststore: rws-client.jks
  truststorePassword: password
```

7. Restart each Interaction Recording Web Services node:

   ```
   systemctl restart gir
   ```

Configuring RWS with Elasticsearch Existing certificates:

1. Make sure you have the Elasticsearch CA certificate file.
2. Create the RWS truststore in .p12 format using keytool:

   ```
   keytool -importcert -trustcacerts -alias elastic-ca -file elastic-stack-ca.crt
   -keystore rws-truststore.p12 -storetype PKCS12
   ```

Elasticsearch Utilities:

1. Below are some useful references from the official Elastic documentation and community resources:
   1. How to enable security, passwords, HTTPS, etc. → Set up minimal security for Elasticsearch
   2. How to enable HTTPS for the REST API. → Configure TLS/SSL on HTTP layer
   3. Details about the elasticsearch-certutil helper script. → Elasticsearch certutil documentation
   4. Which users and roles are already available. → Built-in users and roles

Secure Connections between Elasticsearch Nodes

1. How to secure inter-node communication. → Configure TLS/SSL on Transport layer
2. Edit the elasticsearch.yml according to the configuration. For manual configuration example:

   ```
   xpack.security.transport.ssl.enabled: true
   xpack.security.transport.ssl.keystore.path: /path/to/es-server.jks
   xpack.security.transport.ssl.keystore.password: password
   xpack.security.transport.ssl.truststore.path: /path/to/es-server.jks
   xpack.security.transport.ssl.truststore.password: password
   xpack.security.transport.ssl.verification_mode: certificate
   ```

Cassandra Authentication

Interaction Recording Web Services supports Cassandra authentication, which validates incoming user connections to the Cassandra database. Implementing Cassandra authentication requires you to perform configuration in both Cassandra and Interaction Recording Web Services.

Configure Cassandra Authentication

The user account and password required for authentication are managed inside the **cassandra.yaml** file. Configure Cassandra authentication according to the Cassandra 2.2 documentation.

Interaction Recording Web Services Configuration

To support Cassandra authentication, configure the appropriate credentials in the **cassandraCluster** section of the **application.yaml** file:

```
cassandraCluster:
   thrift_port: 9160
   jmx_port: 7199
   ...
   userName: <superuser name>
   password: <superuser password>
   ...
```

> **Important**
>
> - If the userName and password are not configured, RWS will connect to Cassandra anonymously.
> - To encrypt the password for added security, see Password Encryption.

Password Encryption

For added security, consider encrypting your passwords in the **application.yaml** file by using the following procedure:

1. Run the RWS application with the --**encrypt** parameter followed by the password you need to encrypt. For example, if the password is “ops”:

   ```
   $ java -jar gir.jar --encrypt ops
   CRYPT:an03xPrxLAu9p==
   ```

   RWS will encrypt and print the password. The server will not actually start.

2. Copy the printed encrypted password and paste into the **application.yaml** file. For example:

   ```
   webDAVJksPassword: CRYPT:an03xPrxLAu9p==
   ```

   The server only decrypts passwords that start with the **CRYPT:** prefix. Passwords without the **CRYPT:** prefix are considered plain text and remain unmodified.

CSRF Protection

Interaction Recording Web Services provides protection against Cross Site Request Forgery (CSRF) attacks. For general information and background on CSRF, see the OWASP CSRF Prevention Cheat Sheet.

> **Important**
> If CSRF protection is enabled, then the label/tagging and deletion prevention functionality cannot be used in SpeechMiner, as SpeechMiner does not support CSRF.

To set up Cross Site Request Forgery protection, set the following options in the **serverSettings** section of the **application.yaml** file on each of your Interaction Recording Web Services nodes:

- enableCsrfProtection—determines whether CSRF protection is enabled on the Web Services node.
- crossOriginSettings—specifies the configuration for cross-origin resource sharing in Interaction Recording Web Services. Make sure this option has the **exposedHeaders** setting with a value that includes `X-CSRF-HEADER,X-CSRF-TOKEN`.

For example, your configuration might look like this:

```
enableCsrfProtection: true
crossOriginSettings:
    corsFilterCacheTimeToLive: 120
    allowedOrigins: http://*.genesys.com, http://*.genesyslab.com
    allowedMethods: GET,POST,PUT,DELETE,OPTIONS
    allowedHeaders: "X-Requested-With,Content-Type,Accept,
Origin,Cookie,authorization,ssid,surl,ContactCenterId"
    allowCredentials: true
    exposedHeaders: "X-CSRF-HEADER,X-CSRF-TOKEN"
```

For more information about CSRF protection in the Interaction Recording Web Services API, see Cross Site Request Forgery Protection.

CORS Filter

Interaction Recording Web Services supports Cross-Origin Resource Sharing (CORS) filter, which allows applications to request resources from another domain. For general information and background on CORS, see Cross-Origin Resource Sharing.

To set up Cross-Origin Resource Sharing, make sure you set the crossOriginSettings option in the **serverSettings** section of the **application.yaml** file on each of your Interaction Recording Web Services nodes . It specifies the configuration for cross-origin resource sharing in Interaction Recording Web Services. Make sure this option has the **exposedHeaders** setting with a value that includes `X-CSRF-HEADER,X-CSRF-TOKEN`.

For example, your configuration might look like this:

```
crossOriginSettings:
    corsFilterCacheTimeToLive: 120
    allowedOrigins: http://*.genesys.com, http://*.genesyslab.com
    allowedMethods: GET,POST,PUT,DELETE,OPTIONS
    allowedHeaders: "X-Requested-With,Content-
Type,Accept,Origin,Cookie,authorization,ssid,surl,ContactCenterId,X-CSRF-TOKEN"
    allowCredentials: true
    exposedHeaders: "X-CSRF-HEADER,X-CSRF-TOKEN"
```

For more information about CORS in the Interaction Recording Web Services API, see Cross-Origin Resource Sharing.

Interaction Recording Web Services Authentication Flow

Interaction Recording Web Services provides authentication in the following sequence:

1. Configuration Server Authentication

- If a request contains a basic authentication header and Configuration Server authentication is enabled for this contact center, Configuration Server authentication is applied.
  - If successful, user is authenticated and execution flow proceeds to the authorization stage.
  - If authentication headers are not present, Configuration Server authentication is disabled, or authentication fails, execution flow proceeds to the next step.

2. Interaction Recording Web Services Authentication

- If a request contains a basic authentication header and Configuration Server authentication is not enabled for this contact center, Interaction Recording Web Services authentication is applied.
  - If successful, user is authenticated and execution flow proceeds to the authorization stage.
  - If authentication headers are not present or authentication fails, execution flow proceeds to the next step.

Next Step

- Starting and Testing