Alfresco
Tech Talk Live #157

April 24, 2024
Agenda

- Community news
- Troubleshooting Made Easy: Deciphering Alfresco mTLS Configuration
Alfresco TechQuest

September 10-12, 2024

- Where: Lisbon, Portugal
- Topics:
  - AGS, ADF, APS
  - Out-of-Process extensions
  - Migration: Solr to Elasticsearch & ACS Embedded Workflows to APS
  - Hyland Experience Insight and Automate

Registration is open: https://university.hyland.com/pages/techquest-alfresco

Price

$1250 | $995* EARLY BIRD

Alfresco focused technical conference
September 10-12, 2024 (Lisbon, Portugal)

Register Now

*Early bird pricing available until July 9, 2024
Resources

Alfresco

• Web and API based SMTP testing @GitHub
• Sample space for opensource.hyland.com @ GitHub
• Blog post in Docker page for GenAI Stack @GitHub
• Alfresco mTLS Debugging Kit @GitHub
• GitHub Pages for Alfresco Ansible Deployment @GitHub

Resources to come

• Adapting your logging configuration to log4jv2
• Upgrading your addons to Jakarta EE 10 and Spring 6
• Upgrading to Apache Tomcat 10
• Using Control Center App with Community Edition
• What else?
Contributions

- **OOTBee Support Tools compatible with ACS 23.x** by @Afaust
- **Share action for Alfresco OCR** by @abhinavmishra
- **Starter project for ADF and ACS** by @DenysVuika

Conferences

**OpenSearchCon Europe 2024**, 6-7 May 2024
Berlin, Germany

*Hyland will be exhibitor, swing by and say hello!*
TTL Speakers Wanted!

- Take the opportunity to showcase your work to the community
- About Alfresco, Nuxeo and associated technologies
- Best practices, integration, scaling, cloud, AI…
- In your native language
Today’s Talk
Troubleshooting Made Easy: Deciphering Alfresco mTLS Configuration

Angel Borroy
Developer Evangelist
Hyland
TTL #157
Troubleshooting Made Easy: Deciphering Alfresco’s mTLS Configuration

Angel Borroy
Developer Evangelist
Agenda

• Alfresco mTLS
• Cryptographic Best Practices
• Communication Repository <> Search
• Troubleshooting Tools
• Hands on, using EC certificates
Alfresco mTLS
Alfresco mTLS

Web Proxy

UI

Repository

Transform

Search

DB

keystore

CA

COMMUNITY
Alfresco mTLS

*When using Search Enterprise with Elasticsearch or OpenSearch*
A Closer Look

TLS Protocol

server

TLS Protocol

client

Alfresco Service

keystore TRUST

keystore KEY

CA

Self-Signed Public Authority
Cryptographic Best Practices
General Guidelines

- SSL
- TLSv1.0
- TLSv1.1
- TLSv1.2*
- TLSv1.3

JCEKS
JKS
PKCS12

RSA 2048 bits
ECDSA 224 bits
- Server Authentication
- Client Authentication

OpenSSL
alfresco-ssl-generator
Let’s Encrypt
Use **TLSv1.3**

- Apache Tomcat, set protocols to TLSv1.3 in Connector.SSLHostConfig
- Jetty, set TLSv1.3 in Java property jdk.tls.client.protocols
- Spring Boot, set TLSv1.3 in SERVER_SSL_ENABLED_PROTOCOLS

Alternatively use **TLS 1.2** with *ECDHE* and *AES-GCM* hardcoded

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

When multiple TLS versions are available in the server, the client will select one

- The default for security handshakes in JDK 17 is TLS 1.3
Keystore Type and Certificates

Use Keystore Type **PKCS12**
- Avoid using *non-standard* formats like JKS or JCEKS

**Certificates**
- **Algorithm**
  - **RSA**, widely supported across different platforms and libraries
  - **ECDSA**, equivalent security with shorter key length, more performant and efficient for mTLS
- **Minimum key length**
  - 2048 bits for RSA
  - 224 bits for EC
- **Usage**
  - Server Authentication – OID 1.3.6.1.5.5.7.3.1
  - Client Authentication – OID 1.3.6.1.5.5.7.3.2
Certificate Authority

Self-Signed

• Use Alfresco SSL Generator project, which depends on OpenSSL for certificate generation
• Use alternative software able to issue certificates according with the previous recommendations
  • Later in this session, smallstep will be used

Public Authority

• Use OpenSSL with Let’s Encrypt, set up a cron job to re-fetch certificates regularly
  • Requires active Internet connection to Alfresco containers
• Use a web hosting provider, like AWS
mTLS between Repository and Search
Use `community.sh` script from Alfresco SSL Generator

```
$ ./community.sh
$ tree keystores
keystores
  ├── alfresco
  │   ├── ssl.keystore
  │   └── ssl.truststore
  ├── solr
  │   ├── ssl.repo.client.keystore
  │   └── ssl.repo.client.truststore
  └── client
      └── browser.p12
```

Creating Certificates and Keystores

Solr Admin Web Console
Repository Keystores

$ keytool -v -list -keystore keystores/alfresco/ssl.truststore
Alias name: alfresco.ca
Owner: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US
Issuer: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US
Alias name: ssl.repo.client
Owner: CN=Search, OU=Alfresco, O=Hyland, ST=OH, C=US
Issuer: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US

$ keytool -v -list -keystore keystores/alfresco/ssl.keystore
Alias name: ssl.repo
Owner: CN=Repository, OU=Alfresco, O=Hyland, ST=OH, C=US
Issuer: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US

RSA 2048 bits
• Server Authentication
• Client Authentication
Add following Connector to `${TOMCAT_DIR}/conf/server.xml` file

```xml
    SSLEnabled="true" scheme="https" secure="true"
    defaultSSLHostConfigName="localhost">
    <SSLHostConfig hostName="localhost" protocols="TLSv1.3"
    certificateVerification="required"
    truststoreFile="ssl.truststore"
    truststorePassword="truststore" truststoreType="PKCS12">
        <Certificate certificateKeystoreFile="ssl.keystore"
            certificateKeyAlias="ssl.repo" type="RSA"
            certificateKeystorePassword="keystore"
            certificateKeystoreType="PKCS12"/>
    </SSLHostConfig>
</Connector>
```
Add environment variables containing passwords
-Dssl-keystore.password=keystore
-Dssl-truststore.password=truststore

Set Alfresco Repository Java Properties
solr.host=localhost
solr.port.ssl=8983
solr.secureComms=https
encryption.ssl.keystore.type=PKCS12
encryption.ssl.keystore.location=/usr/local/tomcat/keystore/ssl.keystore
encryption.ssl.truststore.type=PKCS12
encryption.ssl.truststore.location=/usr/local/tomcat/keystore/ssl.truststore

When using the same password for keystore and keys, no aliases setting is required.
Search Keystores

$ keytool -v -list -keystore keystores/solr/ssl.repo.client.truststore
Alias name: ssl.repo
Owner: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US
Issuer: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US
Alias name: alfresco.ca
Owner: CN=Repository, OU=Alfresco, O=Hyland, ST=OH, C=US
Issuer: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US

$ keytool -v -list -keystore keystores/solr/ssl.repo.client.keystore
Alias name: ssl.repo.client
Owner: CN=Search, OU=Alfresco, O=Hyland, ST=OH, C=US
Issuer: CN=Alfresco CA, OU=Alfresco, O=Hyland, L=Cleveland, ST=OH, C=US
Java Environment Variables
- Dsolr.jetty.truststore.password=truststore
- Dsolr.jetty.keystore.password=keystore
- Djdk.tls.client.protocols=TLSv1.3

OS Environment Variables (or modify solr.in.[sh|cmd] file)
SOLR_SSL_KEY_STORE: "/opt/alfresco-search-services/keystore/ssl.repo.client.keystore"
SOLR_SSL_KEY_STORE_PASSWORD: "keystore"
SOLR_SSL_KEY_STORE_TYPE: "PKCS12"
SOLR_SSL_TRUST_STORE: "/opt/alfresco-search-services/keystore/ssl.repo.client.truststore"
SOLR_SSL_TRUST_STORE_PASSWORD: "truststore"
SOLR_SSL_TRUST_STORE_TYPE: "PKCS12"
SOLR_SSL_NEED_CLIENT_AUTH: "true"
Search Configuration (client)

Add environment variables containing passwords
-Dssl-keystore.password=keystore
-Dssl-truststore.password=truststore

Set solrcore.properties Java Properties (in each core or in template)
alfresco.host=localhost
alfresco.port=8443
alfresco.secureComms=https
alfresco.encryption.ssl.keystore.location=/opt/alfresco-search-services/keystore/ssl.repo.client.keystore
alfresco.encryption.ssl.keystore.type=PKCS12
alfresco.encryption.ssl.truststore.location=/opt/alfresco-search-services/keystore/ssl.repo.client.truststore
alfresco.encryption.ssl.truststore.type=PKCS12

When using the same password for keystore and keys, no aliases setting is required.
Sample deployment with Docker Compose

https://github.com/aborroy/alfresco-mtls-debugging-kit/tree/main/docker
Troubleshooting tools
Available tools

- alfresco-http-java-client
- solr-http-java-client
- DB
- Repository
- UI
- Transform
- Search
- Web Proxy

https://github.com/aborroy/alfresco-mtls-debugging-kit
Admin Web Console

Deploy as addon
alfresco-http-java-client.jar
crypto-utils.jar

Source code

App URL
http://localhost:8080/alfresco/s/admin/admin-search-client

Credentials
ADMINISTRATOR, default admin/admin
Alfresco Search Services

Solr REST API Action

Deploy as plugin
alfresco-http-java-client.jar
solr-http-java-client.jar
config/solr.xml
Source code

App URL
https://localhost:8983/solr/admin/cores?action=HTTP-CLIENT&coreName=alfresco

Credentials
Client certificate, like browser.p12
Run as program

```sh
$ java -jar target/mtls-conf-app.jar
```

**ERRORS for ENDPOINT:**

Current truststore seems to be wrong. It does not include TRUST certificates provided by the endpoint.

**ERRORS DETAIL:**

```
sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target
```

**Source code**

Hands on, using EC certificates
Use ECC 256 bits certificates for ECDSA with **step-ca**

- Around *10% faster* than RSA 2048 bits for Alfresco mTLS
- Less bandwidth consumption
- Higher security (as RSA 2048 is equivalent to ECC 224)


```bash
# Start step-ca container, default CA will be created in “step” folder
$ docker compose up

# Install step CLI
$ brew install step

# Get CA password
$ cat step/secrets/password
ZuSJLBo6uRtlvzGe0z1i5ReqU2tpncl19RBUIf5V```
# Create a certificate for alfresco, use “keystore” password to protect the key
$ step certificate create alfresco alfresco.crt alfresco.key \
  --profile leaf --not-after=8760h --bundle --ca step/certs/root_ca.crt \
  --ca-key step/secrets/root_ca_key

# Create a certificate for solr, use “keystore” password to protect the key
$ step certificate create solr solr.crt solr.key \
  --profile leaf --not-after=8760h --bundle --ca step/certs/root_ca.crt \
  --ca-key step/secrets/root_ca_key
# Build Keystore and Truststore for alfresco

```bash
$ openssl pkcs12 -export -in alfresco.crt -inkey alfresco.key \
-out alfresco.pkcs12 -name alfresco -noiter -nomaciter
$ keytool -import -alias solr -file solr.crt -keystore \
alfresco-truststore.pkcs12 -storetype PKCS12 -storepass truststore
$ keytool -import -alias ca -file step/certs/root_ca.crt -keystore \
alfresco-truststore.pkcs12 -storetype PKCS12 -storepass truststore
```

# Build Keystore and Truststore for solr

```bash
$ openssl pkcs12 -export -in solr.crt -inkey solr.key \
-out solr.pkcs12 -name solr -noiter -nomaciter
$ keytool -import -alias alfresco -file alfresco.crt -keystore \
solr-truststore.pkcs12 -storetype PKCS12 -storepass truststore
$ keytool -import -alias ca -file step/certs/root_ca.crt -keystore \
solr-truststore.pkcs12 -storetype PKCS12 -storepass truststore
```
# Get alfresco client certificate to access Solr (or re-use alfresco.pkcs12)

$ openssl pkcs12 -export -out browser.p12 -inkey alfresco.key -in alfresco.crt

# Modify compose.yaml to use the new keystores

**Alfresco**
- keystore=alfresco.pkcs12
- truststore=alfresco-truststore.pkcs12
- cert-alias=alfresco
- cert-type=EC

**Solr**
- keystore=solr.pkcs12
- truststore=solr-truststore.pkcs12
# Bonus verification

```sh
$ nmap --script ssl-enum-ciphers -p 8983 localhost
|   TLSv1.3:
|     ciphers:
|     TLS_AKE_WITH_AES_256_GCM_SHA384 (secp256r1)
|     TLS_AKE_WITH_AES_128_GCM_SHA256 (secp256r1)
|     TLS_AKE_WITH_CHACHA20_POLY1305_SHA256 (secp256r1)
```

```sh
$ nmap --script ssl-enum-ciphers -p 8443 localhost
|   TLSv1.3:
|     ciphers:
|     TLS_AKE_WITH_AES_128_CCM_SHA256 (ecdh_x25519)
|     TLS_AKE_WITH_AES_128_GCM_SHA256 (ecdh_x25519)
|     TLS_AKE_WITH_AES_256_GCM_SHA384 (ecdh_x25519)
|     TLS_AKE_WITH_CHACHA20_POLY1305_SHA256 (ecdh_x25519)
```
Thanks!