Tips for Hardening TLS for CDH Services

Created by Chee Kian Koh, last modified on 03/27/2018

This How-To article objective is to gather as much information as possible on how to configure and/or hack CDH to harden TLS for the CDH services. This will hopefully help to save some pain for our SA/SC who needs to do this at a customer and don't have to start researching from scratch. If the cluster that you are hardening is CDH 5.13.x or higher, then you will be glad most of the TLS customization is already supported via safety valves. However, if you are doing it for older CDH version, then tough luck, as not all of the CDH components support chang TLS ciphers and protocol (e.g. Impala). There are some workarounds that may require modifying of the CDH scripts but consider all the risks if you have to apply these in your customer er and ensure customer is aware of the risks!

When hardening TLS, generally there are two things to consider - the ciphers and the TLS protocol to enable/disable.

> This wiki guide is meant to be a live document that will need to be update constantly when new versions are released. Anyone please feel free to update this page if there are any ne outdated information here. 🙂

Services Based On Java (Including Hive, HDFS, YARN, CM, Oozie, Hbase)

Modify java.security policies

For those services that are based on Java, what you need to do is to disable the ciphers on Java which will then applied to most of the Java-based services once they are restarted. Edit th java.security policy file located in $JAVA_HOME/jre/lib/security directory and configure the setting jdk.tls.disabledAlgorithms. The example property in the policy file only disables SSLv3, M cipher suites and enforcing RSA key length to at least 2048 bits. Change the property to the following:

$JAVA_HOME/jre/lib/security/java.security

```
#
# Example:
#   jdk.tls.disabledAlgorithms=MD5, SSLv3, DSA, RSA keySize < 2048
jdk.tls.disabledAlgorithms=TLSv1, TLSv1.1, SHA1, DESede, DES, SSLv3, RC4, MD5withRSA, DH keySize < 2048, EC keySize < 224
```

The cipher suites that I am disabling are:

- Ciphers from TLSv1.0 and TLSv1.1
- Ciphers suite SHA1, DES/3DES, MD5, RC4
- Enforce key length for DH and EC cryto

Below shows the enabled TLS ciphers before and after the change:

Enabled ciphers before hardening	Enabled ciphers after hardening

Enabled ciphers before hardening	Enabled ciphers after hardening

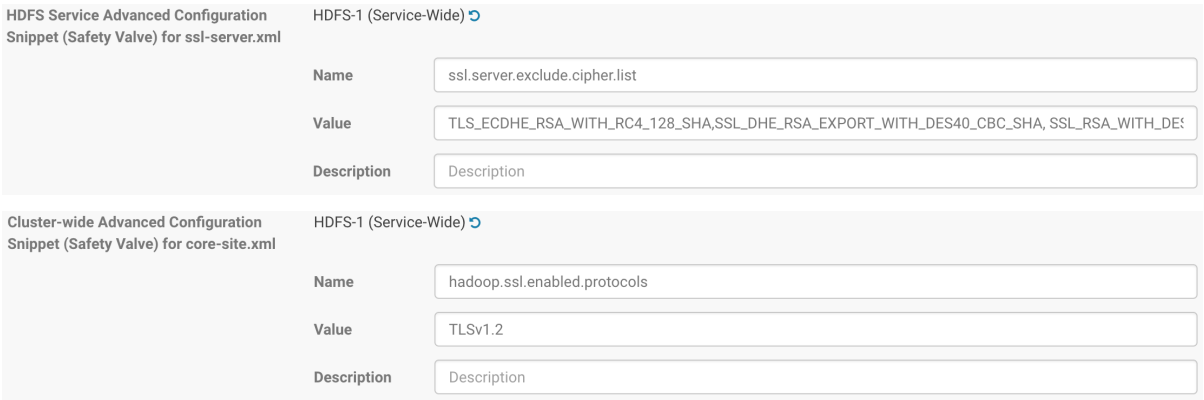
```
| ssl-enum-ciphers:
| TLSv1.0:
| ciphers:
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256k1) - A
| TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
| TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
| TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256k1) - C
| TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C
| compressors:
| NULL
| cipher preference: server
| warnings:
| 64-bit block cipher 3DES vulnerable to SWEET32 attack
| Key exchange (dh 1024) of lower strength than certificate key
| TLSv1.1:
| ciphers:
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256k1) - A
| TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
| TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
| TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256k1) - C
| TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C
| compressors:
| NULL
| cipher preference: server
| warnings:
| 64-bit block cipher 3DES vulnerable to SWEET32 attack
| Key exchange (dh 1024) of lower strength than certificate key
| TLSv1.2:
| ciphers:
| TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (dh 1024) - A
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256k1) - C
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256k1) - A
| TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C
| TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A
| TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A
| TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
| TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
| TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A
| TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A
| compressors:
| NULL
| cipher preference: client
| warnings:
| 64-bit block cipher 3DES vulnerable to SWEET32 attack
| Key exchange (dh 1024) of lower strength than certificate key
|_ least strength: D
```

```
| ssl-enum-ciphers:
| TLSv1.2:
| ciphers:
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (dh 1024) - A
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (dh 1024) - A
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (secp256k1) - A
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256k1) - A
| TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A
| TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
| TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A
| TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A
| compressors:
| NULL
| cipher preference: client
| warnings:
| Key exchange (dh 1024) of lower strength than certificate key
|_ least strength: A
```

Hadoop settings in core-site.xml

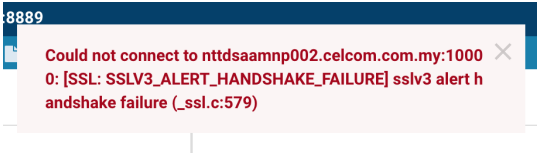
In addition to the TLS paremeters in java.security policy file, you may also need to configure safety valve to update *hadoop.ssl.enabled.protocols* in core-site.xml and *hadoop.ssl.exclude.cipher.suites* in ssl-server.xml. For example, MR encrypted shuffle TLS seems to only take effect from these two parameters even after you have set the TLS paramete
java.security.

Note that these parameters will only apply to HDFS services. You will still need to modify the Java Security policies file described as above for the other CDH services.



Note - Hue SSL Handshake Error

If you are using CDH 5.12.x and earlier and have disabled TLS1.0, you may encounter problem with Hue not able to communicate some of the CDH services after the hardening. For exan
encountered that Hue will fail to communicate with HiveServer2 and threw a SSLV3_ALERT_HANDSHAKE_FAILURE error. This is because by default, Hue is configured to use TLS1.0 a
fail to negotiate a common cipher if we have disabled TLS1.0 on HiveServer2.



To resolve this, please see the Hue section below.

Hue - Disable Weak Ciphers and TLS 1.0/1.1

Hue allows configuration of the cipher list in the hue.ini file. We can configure this in Cloudera Manager using safety valve.

In Cloudera Manager, go to Hue configuration, then **Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini**, and add **ssl_cipher_list** as per the followi

```
[desktop]
ssl_cipher_list=ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS:!3DES:!DES:!SHA
[[session]]
secure=true
http-only=true
expire_at_browser_close=true
ttl=86400
```

While you are at it, you should also add the rest of the **session** settings to the safety valve as well, as these are likely to be flagged out as security risks:

secure=true	Enable tagging of secure flag to the users' session ID cookie. Prevents man-in-middle attack.
http-only=true	Enable tagging of http-only flag to the users' session ID cookie. Prevents cross-site scripting vulnerability.
expire_at_browser_close=true	Terminate Hue logon session after user closed the browser.
ttl=86400	Expire the users' session ID cookie after 1 day (24 hours).

The safety valve above allows you to specify which ciphers to enable. However, if you need to disable a particular TLS version explicitly, you will need to modify the WSGIserver python sc uses. The python file is located at /opt/cloudera/parcels/CDH/lib/hue/desktop/core/src/desktop/lib/wsgiserver.py.

wsgiserver.py

```
1699  ctx.set_cipher_list(self.ssl_cipher_list)
1700  try:
1701    ctx.use_privatekey_file(self.ssl_private_key)
1702    ctx.use_certificate_file(self.ssl_certificate)
1703    if self.ssl_certificate_chain:
1704      ctx.use_certificate_chain_file(self.ssl_certificate_chain)
1705  except Exception, ex:
1706    logging.exception('SSL key and certificate could not be found or have a problem')
1707    raise ex
1708  ctx.set_options(SSL.OP_NO_SSLv2 | SSL.OP_NO_SSLv3)
1709  self.socket = SSLConnection(ctx, self.socket)
1710  self.populate_ssl_environ()
```

Change the line 1708 to from:

ctx.set_options(SSL.OP_NO_SSLv2 I SSL.OP_NO_SSLv3)

to

ctx.set_options(SSL.OP_NO_SSLv2 I SSL.OP_NO_SSLv3 **I SSL.OP_NO_TLSv1)**

Note: don't try to use SSL.OP_NO_TLSv1_1 in the options as Hue will complain that this option does not exists and will fail to start. From what I know, you need to upgrade certain python make this work.

> Modifying any of our product code or script is not recommended. Do it at your own risk!

Here is the enabled ciphers for Hue before and after the changes:

Enabled ciphers before hardening	Enabled ciphers after hardening

Enabled ciphers before hardening	Enabled ciphers after hardening
8888/tcp open sun-answerbook | ssl-enum-ciphers: | TLSv1.0: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A | compressors: | NULL | cipher preference: client | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | TLSv1.1: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A | compressors: | NULL | cipher preference: client | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | TLSv1.2: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A | compressors: | NULL | cipher preference: client | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack |_ least strength: C	8888/tcp open sun-answerbook | ssl-enum-ciphers: | TLSv1.2: | ciphers: | TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A | compressors: | NULL | cipher preference: client |_ least strength: A

Hue Support for TLS1.1 and TLS1.2 Thrift Connection

If you have disabled TLS1.0 for HiveServer2, you may get an error when Hue is trying to connect to the Thrift service if you are deploying on CDH 5.12 or earlier. This is because Hue's implementation of Thrift is hardcoded to TLS 1.0 and no other version of TLS. To workaround this, you need to hack the python script /opt/cloudera/parcels/CDH/lib/hue/build/env/lib/python packages/thrift-0.9.1-py2.7-linux-x86_64.egg/thrift/transport/TSSLSocket.py (for CDH5.12 only, path to the file may be different for older versions).

TSSLSocket.py

```
35  The protocol used is set using the class variable
36  SSL_VERSION, which must be one of ssl.PROTOCOL_* and
37  defaults to  ssl.PROTOCOL_TLSv1 for greatest security.
38  """
39  SSL_VERSION = ssl.PROTOCOL_TLSv1
40
41  def __init__(self,
42               host='localhost',
43               port=9090,
44               validate=True,
45               ca_certs=None,
46               keyfile=None,
47               certfile=None,
48               unix_socket=None):
```

Change line 39 from ssl.PROTOCOL_TLSv1 to ssl.PROTOCOL_SSLv23 (which will allow negotatiation of TLS version).

Note that this is already fixed for CDH 5.13 (see fixed issue for CDH5.13 - Issue HUE-7155).

Hue Load Balancer

Hue Load Balancer works as a reverse proxy to the Hue service and it has its own TLS cipher configurations. It is based on Apacher web server, so the SSL/TLS configuration is located in at Hue's */var/run/cloudera-scm-agent/process* directory. If your CDH is 5.13, then you can customize the TLS ciphers by HUE Load Balancer Advanced Configuration Snippet (httpd.conf s The settings for the safety valve configuration are SSLProtocol and SSLCipherSuite.

Example:

```
SSLProtocol TLSv1.2
SSLCipherSuite ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS:!3DES:!DES:!SHA
```

Reference:

- ~~OPSAPS-32597~~ - Disabling certain cyphers in HUE load balancer RESOLVED
- ~~OPSAPS-42187~~ - [Hue] Enable TLS v1.2 setting for Hue Load Balancer RESOLVED

For CDH version that is earlier than 5.13.x, unfortunately the safety valve will not work. A workaround is to hack the CM script that generates the httpd.conf located in */var/run/cloudera-scm agent/process* directory to replace the two SSLProtocol and SSLCipherSuite settings with the customized SSL settings. Modify the script located at */usr/lib64/cmf/service/hue/httpd.sh* and 134 and 135 with the appropriate perl regex.

Example from CDH 5.12

/usr/lib64/cmf/service/hue/httpd.sh

```
perl -pi -e "s#{{CLOUDERA_HTTPD_CONF_DIR}}#$CLOUDERA_HTTPD_CONF_DIR#g" $CLOUDERA_HTTPD_CONF_DIR/*.conf
perl -pi -e "s#{{CLOUDERA_HTTPD_LOG_DIR}}#$CLOUDERA_HTTPD_LOG_DIR#g" $CLOUDERA_HTTPD_CONF_DIR/*.conf
perl -pi -e "s#{{CLOUDERA_HTTPD_MODULE_DIR}}#$CLOUDERA_HTTPD_MODULE_DIR#g" $CLOUDERA_HTTPD_CONF_DIR/*.conf
perl -pi -e "s#all -SSLv2 -SSLv3#TLSv1.2#g" $CLOUDERA_HTTPD_CONF_DIR/*.conf
perl -pi -e "s#ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA3
```

Modifying any of our product code or script is not recommended. Do it at your own risk!

Impala

Impala has a flag which we can use to configure the cipher list to enable/disable. However, this flag is only added for Impala 2.10 which is packaged with CDH 5.13. The JIRA issue for this enhancement is IMPALA-5696 and it is currently not backported to the earlier releases. So if you are using CDH 5.12 and earlier, then this will not apply and use some other options like us balancer to front the Impalad and restrict the TLS ciphers on the load balancer. This means you need to use TLS proxy rather than passthrough. More details in the HAProxy section.

From https://www.cloudera.com/documentation/enterprise/latest/topics/impala_ssl.html#tls_min_version:

Specify one of the following values for the --ssl_minimum_version configuration setting:

- tlsv1: Allow any TLS version of 1.0 or higher. This setting is the default when TLS/SSL is enabled.
- tlsv1.1: Allow any TLS version of 1.1 or higher.
- tlsv1.2: Allow any TLS version of 1.2 or higher.

Along with specifying the version, you can also specify the allowed set of TLS ciphers by using the --ssl_cipher_list configuration setting. The argument to this option is a list of keywords, s by colons, commas, or spaces, and optionally including other notation.

To add this flag using Cloudera Manager, basically add it as an safety valve **Impala Daemon Command Line Argument Advanced Configuration Snippet (Safety Valve)**:

```
--ssl_minimum_version=tlsv1.2
--ssl_cipher_list=ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS:!3DES:!DES:!SHA
```

Reference:

IMPALA-5696 - Enable cipher configuration when using TLS w/Thrift

IMPALA-5743 - Allow for configuration of TLS / SSL versions

Here is the list of ciphers enabled on Impalad before and after the safety valve (port 21000 and 21050 has the same ciphers):

Enabled ciphers before hardening	Enabled ciphers after hardening

Enabled ciphers before hardening	Enabled ciphers after hardening
21000/tcp open irtrans | ssl-enum-ciphers: | TLSv1.0: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_IDEA_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_RC4_128_MD5 (rsa 2048) - C | TLS_RSA_WITH_RC4_128_SHA (rsa 2048) - C | TLS_RSA_WITH_SEED_CBC_SHA (rsa 2048) - A | compressors: | NULL | cipher preference: client | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | 64-bit block cipher IDEA vulnerable to SWEET32 attack | Broken cipher RC4 is deprecated by RFC 7465 | Ciphersuite uses MD5 for message integrity | TLSv1.1: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_IDEA_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_RC4_128_MD5 (rsa 2048) - C | TLS_RSA_WITH_RC4_128_SHA (rsa 2048) - C | TLS_RSA_WITH_SEED_CBC_SHA (rsa 2048) - A | compressors: | NULL | cipher preference: client | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | 64-bit block cipher IDEA vulnerable to SWEET32 attack | Broken cipher RC4 is deprecated by RFC 7465 | Ciphersuite uses MD5 for message integrity | TLSv1.2: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 2048) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_IDEA_CBC_SHA (rsa 2048) - A | TLS_RSA_WITH_RC4_128_MD5 (rsa 2048) - C | TLS_RSA_WITH_RC4_128_SHA (rsa 2048) - C | TLS_RSA_WITH_SEED_CBC_SHA (rsa 2048) - A | compressors: | NULL | cipher preference: client | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | 64-bit block cipher IDEA vulnerable to SWEET32 attack | Broken cipher RC4 is deprecated by RFC 7465 | Ciphersuite uses MD5 for message integrity |_ least strength: C	21000/tcp open irtrans | ssl-enum-ciphers: | TLSv1.2: | ciphers: | TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A | TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A | compressors: | NULL | cipher preference: client |_ least strength: A

Impala Shell Support for TLS1.1 and TLS1.2

For CDH version that is earlier than 5.13.x, impala-shell is hardcoded to use TLS1.0 and will fail to connect if the backend service if using TLS1.1/1.2, for example if connecting through a L Balancer that has hardened TLS. This is fixed already if you are using CDH5.13 which updated the Impala-shell python script to allow the shell to negotiate the TLS protocol based on the service. If you are not using CDH5.13.x or higher, you can check out JIRA IMPALA-5775 for what modifications to the shell python script to support higher TLS protocol.

Reference:

IMPALA-5775 - Impala shell only supports TLSv1

HAProxy TLS Hardening

Usually when we use a load balancer to load balance Hive and Impala with TLS enabled, most of the guides and documentations use SSL/TLS passthrough. In this case, whatever ciphers protocol that you have enabled/disabled on the backend Hive or Impala services should passthrough to the client. However, most load balancer can also be configured as SSL/TLS proxy w connections terminate on the load balancer. This is useful for scenario where changing the ciphers or TLS protocol is not supported (e.g. Impala on older CDH), you can minimize the secu only exposing the service externally via a load balancer with hardened TLS configuration.

A common scenario would be to deploy HAProxy as TLS Proxy and customize the TLS settings. Below is a sample haproxy.cfg as a reference, the SSL/TLS hardening settings are on line

```

#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     10000
    user        haproxy
```

```
    group       haproxy
    daemon

    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

    # configure ssl/tls
    tune.ssl.default-dh-param 2048
    ssl-default-bind-options no-sslv3 no-tlsv10 no-tlsv11
    ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS:!

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          15m
    timeout server          15m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 10000

#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------

frontend impala-21000
    mode                tcp
    bind                0.0.0.0:21000 ssl crt /etc/haproxy/haproxy.pem
    default_backend     impala-21000-backend

frontend impala-21050
    mode                tcp
    bind                0.0.0.0:21050 ssl crt /etc/haproxy/haproxy.pem
    default_backend     impala-21050-backend

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------
backend impala-21000-backend
    mode        tcp
    balance     roundrobin
    server              impala01        impala01.domain.com:21000       check ssl verify none
    server              impala02        impala02.domain.com:21000       check ssl verify none

backend impala-21050-backend
    mode        tcp
    balance     roundrobin
    server              impala01        impala01.domain.com:21050       check ssl verify none
    server              impala02        impala02.domain.com:21050       check ssl verify none
```

Other References

- SFDC - How to enforce TLS 1.2 in the Enterprise Data Hub (EDH)