Customize the Expressway SSL Cipher Configuration

Contents

Introduction

Prerequisites

- Requirements
- Components Used

Background Information

- Inspect the Cipher String
- Inspect the Cipher Negotiation in the TLS Handshake with a Packet Capture

Configure

- Disable a Specific Cipher
- Disable a Group of Ciphers Using a Common Algorithm

Verify

- Inspect the List of Ciphers Allowed by the Cipher String
- Test a TLS Connection by Negotiating a Disabled Cipher
- Inspect a Packet Capture of a TLSHandshake Using a Disabled Cipher

Related Information

Introduction

This document describes the steps to customize the preconfigured cipher strings on Expressway.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- Cisco Expressway or Cisco VCS.
- TLS protocol.

Components Used

The information in this document is based on these software and hardware versions:

- Cisco Expressway version X15.0.2.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

The default Expressway configuration includes preconfigured cipher strings, which for compatibility reasons, enable support for some ciphers that can be considered weak under some enterprise security policies. It is possible to customize the cipher strings in order to fine tune them to fit the specific policies of each environment.

In Expressway, it is possible to configure an independent cipher string for each of these protocols:

- HTTPS
- LDAP
- Reverse proxy
- SIP
- SMTP
- TMS provisioning
- UC server discovery
- XMPP

The cipher strings obey the OpenSSL format described in the OpenSSL Ciphers Manpage. The current Expressway version X15.0.2 comes with the default string EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH preconfigured for all protocols equally. From the web admin page, under **Maintenance > Security > Ciphers**, you can modify the cipher string assigned to each protocol, in order to add or remove specific ciphers or groups of ciphers using a common algorithm.

Inspect the Cipher String

By using the **openssl ciphers -V "<cipher string>"** command, you can output a list with all of the ciphers that a certain string allows, which is useful for visually inspecting the ciphers. This example shows the output when inspecting the default Expressway cipher string:

<#root>

~ #

openssl ciphers -V "EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH"

```
0x13,0x02 - TLS_AES_256_GCM_SHA384 TLSv1.3 Kx=any Au=any Enc=AESGCM(256) Mac=AEAD
0x13,0x03 - TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any Au=any Enc=CHACHA20/POLY1305(256) Mac=AEAD
0x13,0x01 - TLS_AES_128_GCM_SHA256 TLSv1.3 Kx=any Au=any Enc=AESGCM(128) Mac=AEAD
0xC0,0x2C - ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(256) Mac=AEAD
0xC0,0x30 - ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(256) Mac=AEAD
0xCC,0xA9 - ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
0xCC,0xA8 - ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
0xC0,0xAD - ECDHE-ECDSA-AES256-CCM TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESCCM(256) Mac=AEAD
0xC0,0x2B - ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(128) Mac=AEAD
0xC0,0x2F - ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(128) Mac=AEAD
0xC0,0xAC - ECDHE-ECDSA-AES128-CCM TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESCCM(128) Mac=AEAD
0xC0,0x24 - ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA384
0xC0,0x28 - ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA384
0xC0,0x23 - ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA256
0xC0,0x27 - ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA256
0xC0,0x09 - ECDHE-ECDSA-AES128-SHA TLSv1 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA1
0xC0,0x13 - ECDHE-RSA-AES128-SHA TLSv1 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA1
0x00,0xA3 - DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=DSS Enc=AESGCM(256) Mac=AEAD
0x00,0x9F - DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=RSA Enc=AESGCM(256) Mac=AEAD
0xCC,0xAA - DHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=DH Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
0xC0,0x9F - DHE-RSA-AES256-CCM TLSv1.2 Kx=DH Au=RSA Enc=AESCCM(256) Mac=AEAD
0x00,0xA2 - DHE-DSS-AES128-GCM-SHA256 TLSv1.2 Kx=DH Au=DSS Enc=AESGCM(128) Mac=AEAD
```

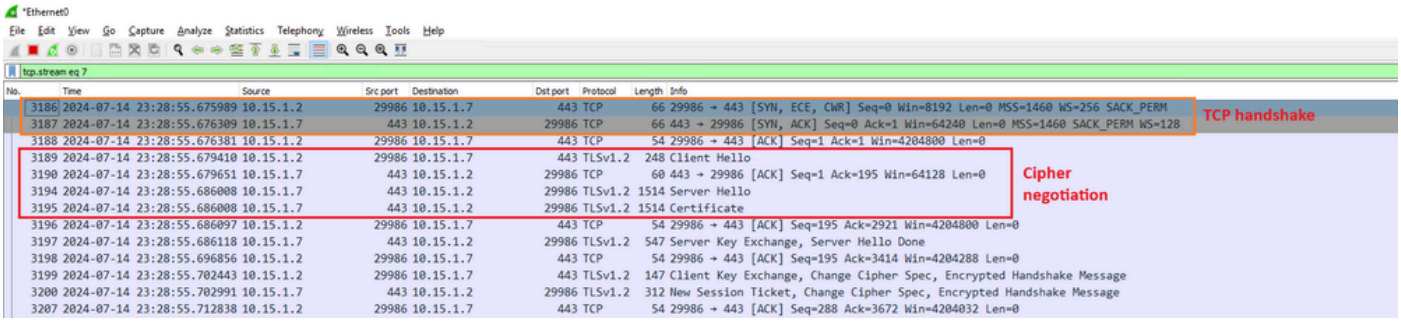
```
0x00,0x9E - DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AESGCM(128) Mac=AEAD
0xC0,0x9E - DHE-RSA-AES128-CCM TLSv1.2 Kx=DH Au=RSA Enc=AESCCM(128) Mac=AEAD
0x00,0x6B - DHE-RSA-AES256-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AES(256) Mac=SHA256
0x00,0x6A - DHE-DSS-AES256-SHA256 TLSv1.2 Kx=DH Au=DSS Enc=AES(256) Mac=SHA256
0x00,0x67 - DHE-RSA-AES128-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AES(128) Mac=SHA256
0x00,0x40 - DHE-DSS-AES128-SHA256 TLSv1.2 Kx=DH Au=DSS Enc=AES(128) Mac=SHA256
0x00,0x33 - DHE-RSA-AES128-SHA SSLv3 Kx=DH Au=RSA Enc=AES(128) Mac=SHA1
0x00,0x32 - DHE-DSS-AES128-SHA SSLv3 Kx=DH Au=DSS Enc=AES(128) Mac=SHA1
0x00,0x9D - AES256-GCM-SHA384 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(256) Mac=AEAD
0xC0,0x9D - AES256-CCM TLSv1.2 Kx=RSA Au=RSA Enc=AESCCM(256) Mac=AEAD
0x00,0x9C - AES128-GCM-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(128) Mac=AEAD
0xC0,0x9C - AES128-CCM TLSv1.2 Kx=RSA Au=RSA Enc=AESCCM(128) Mac=AEAD
0x00,0x3D - AES256-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA256
0x00,0x3C - AES128-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(128) Mac=SHA256
0x00,0x2F - AES128-SHA SSLv3 Kx=RSA Au=RSA Enc=AES(128) Mac=SHA1
~ #
```

Inspect the Cipher Negotiation in the TLS Handshake with a Packet Capture

By capturing a TLS negotiation in a packet capture, you can inspect the details of the cipher negotiation by using Wireshark.

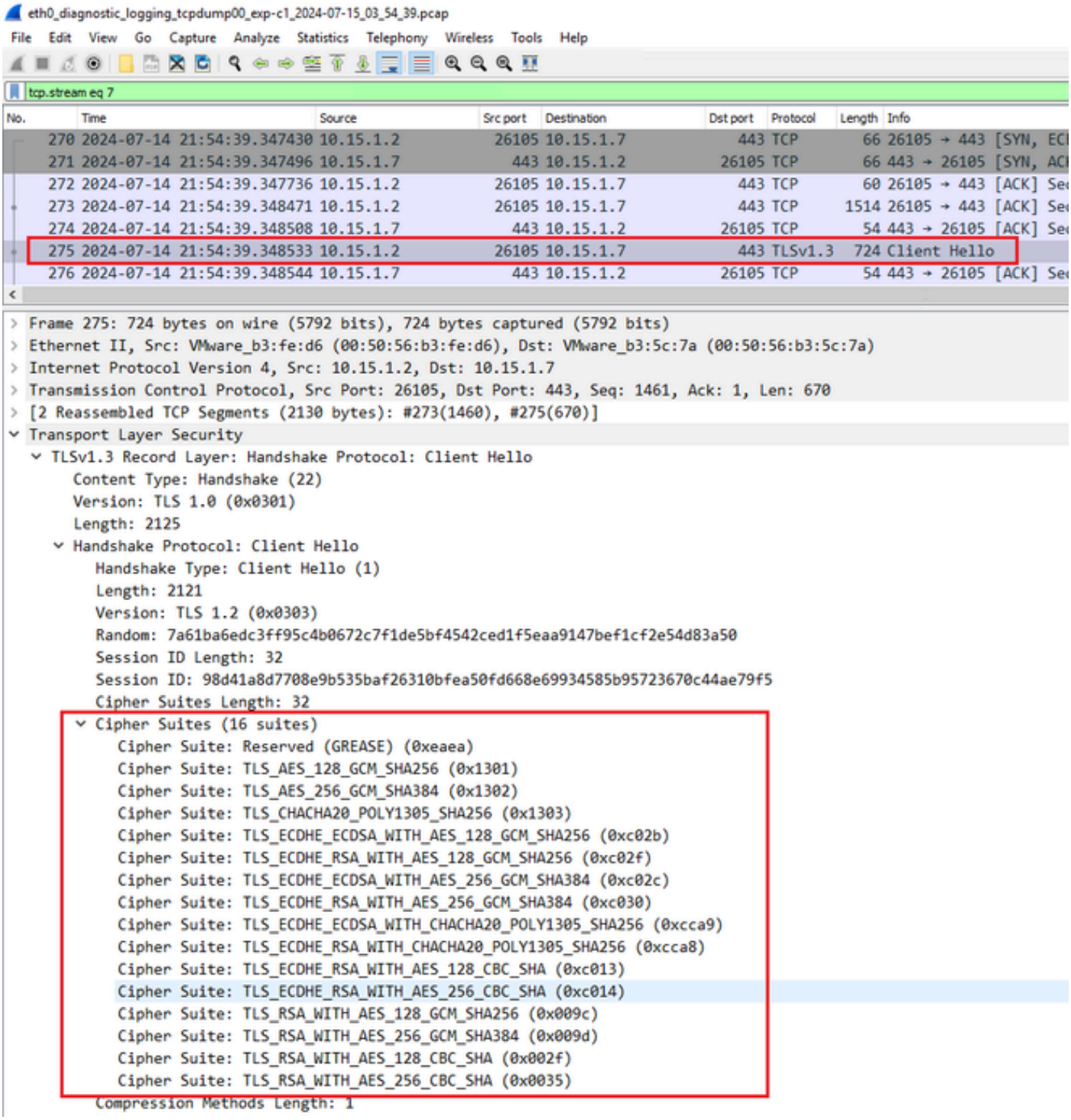
The TLS handshake process includes a ClientHello packet sent by the client device, providing the list of the ciphers it supports according to its configured cipher string for the connection protocol. The server reviews the list, compares it with its own list of allowed ciphers (determined by its own cipher string), and chooses a cipher that both systems support, to be used for the encrypted session. Then it responds with a ServerHello packet indicating the chosen cipher. There are important differences between the TLS 1.2 and 1.3 handshake dialogs, however the cipher negotiation mechanism uses this same principle in both versions.

This is an example of a TLS 1.3 cipher negotiation between a web browser and Expressway on port 443 as seen in Wireshark:



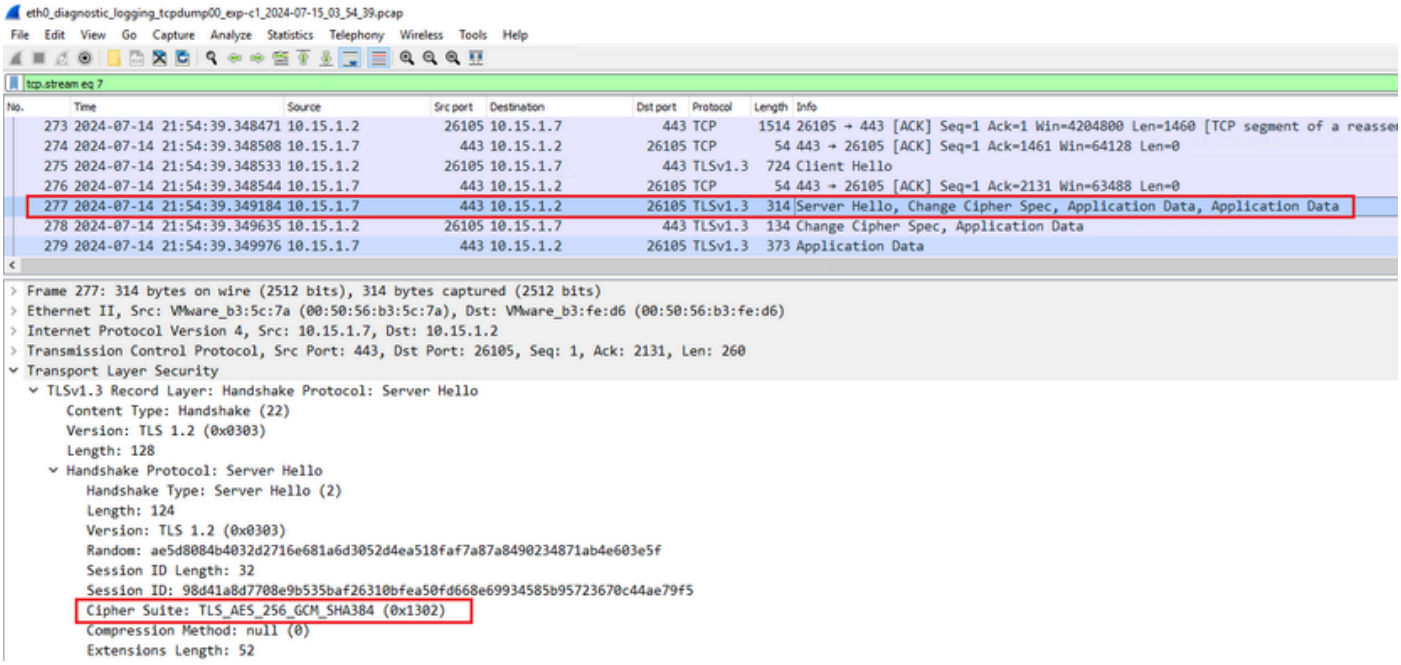
Example of a TLS Handshake in Wireshark

First, the browser sends a ClientHello packet with the list of ciphers it supports:



Example of a ClientHello Packet in Wireshark

Expressway checks its cipher string configured for the HTTPS protocol, and finds a cipher that both itself and the client support. In this example the ECDHE-RSA-AES256-GCM-SHA384 cipher is selected. Expressway responds with its ServerHello packet indicating the selected cipher:



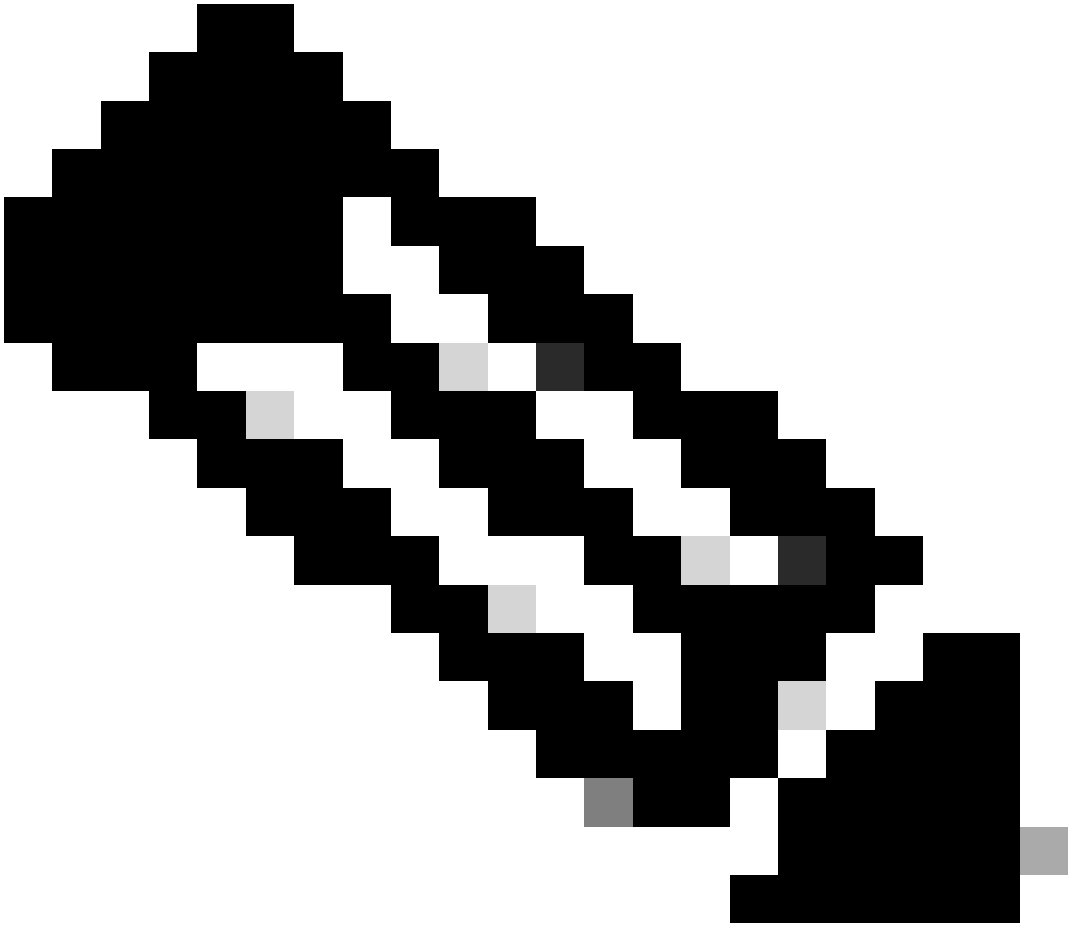
Example of a ServerHello Packet in Wireshark

Configure

The OpenSSL cipher string format includes several special characters in order to perform operations on the string such as removing a specific cipher or a group of ciphers sharing a common component. Since the objective of these customizations is usually removing ciphers, the characters used in these examples are:

- The **-** character, used to remove ciphers from the list. Some or all of the removed ciphers can be allowed again by options appearing later in the string.
- The **!** character, also used to remove ciphers from the list. When using it, the removed ciphers cannot be allowed again by any other options appearing later in the string.
- The **:** character, which acts as the separator between items in the list.

Both can be used to remove a cipher from the string, however **!** is preferred. For a complete list of special characters, review the OpenSSL Ciphers Manpage.



Note: The OpenSSL site states that when using the **!** character, "the ciphers deleted can never reappear in the list even if they are explicitly stated". This does not mean that the ciphers are deleted permanently from the system, it refers to the scope of the interpretation of the cipher string.

Disable a Specific Cipher

In order to disable a specific cipher, append to the default string the **:** separator, the **!** or **-** sign, and the cipher name to be disabled. The cipher name must obey the OpenSSL naming format, available in the OpenSSL Ciphers Manpage. For example, if you need to disable the AES128-SHA cipher for SIP connections, configure a cipher string like this:

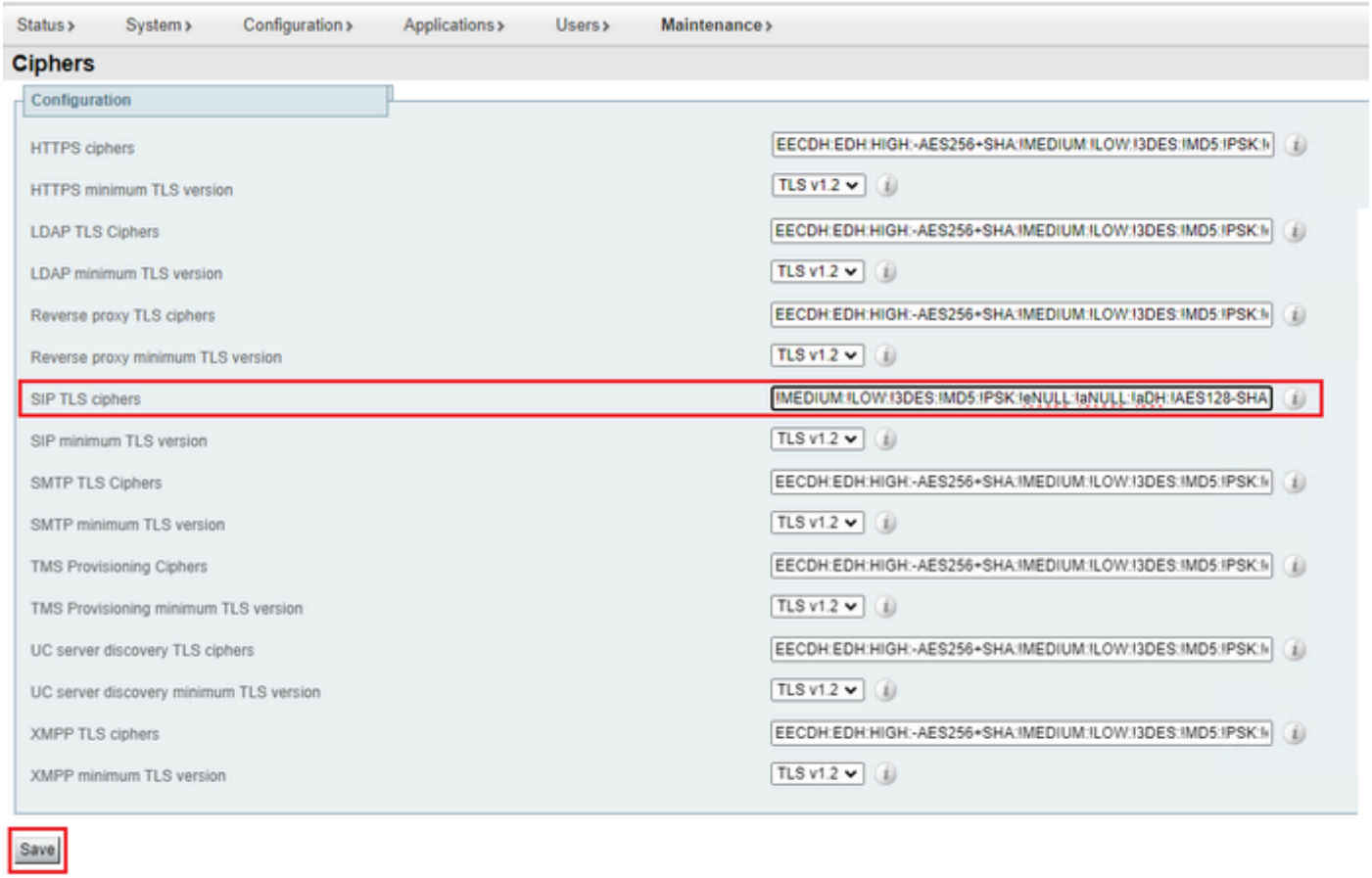
<#root>

```
EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH
```

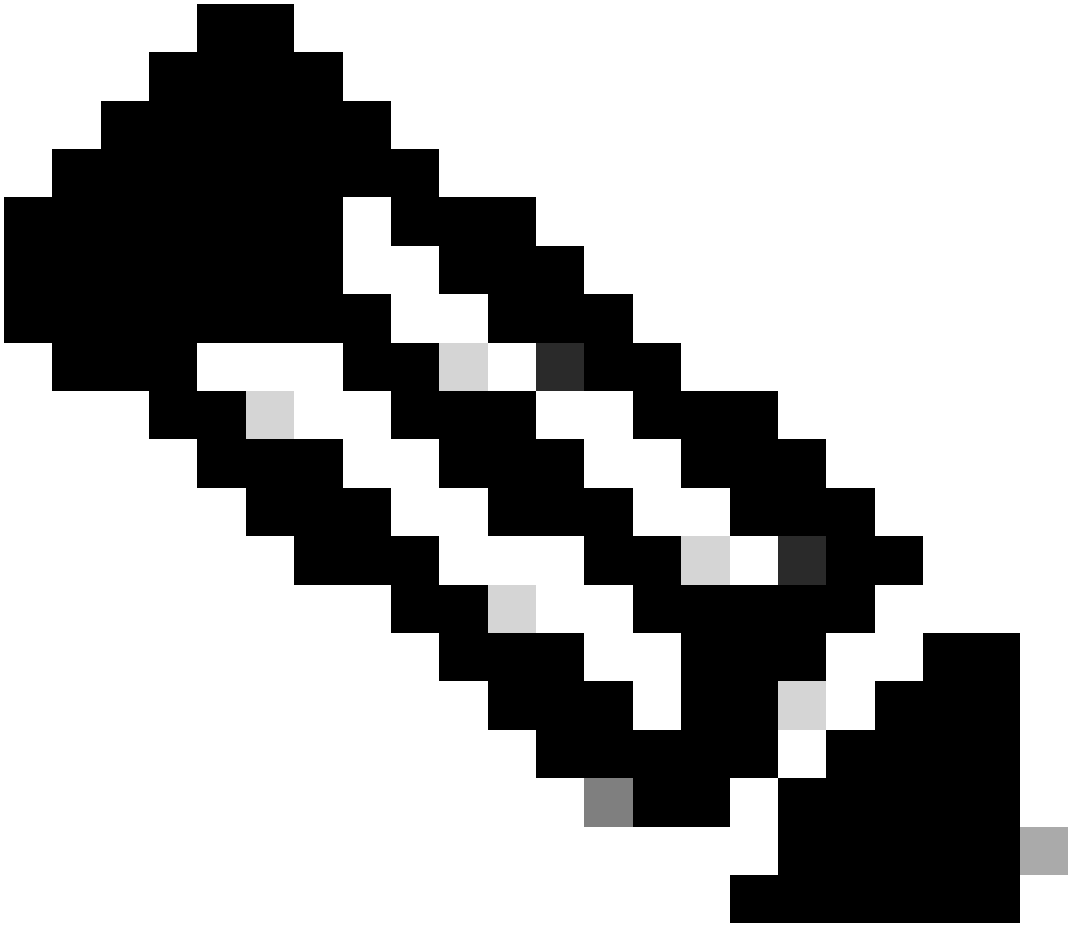
```
:!AES128-SHA
```

Then, navigate to the **Expressway web admin** page, navigate to **Maintenance > Security > Ciphers**,

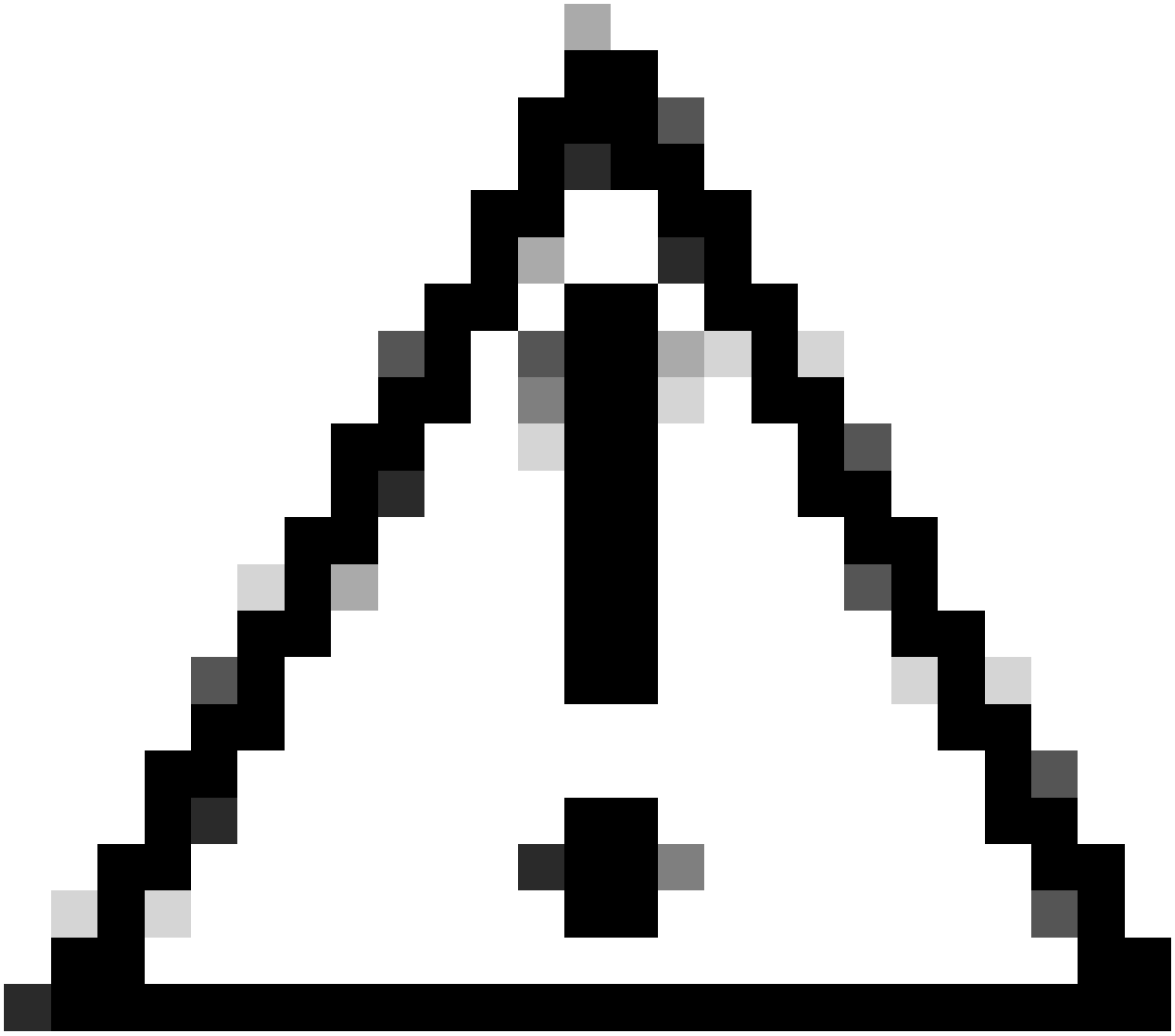
assign the custom string to the required protocol(s), and click **Save**. For the new configuration to be applied, a system restart is required. In this example, the custom string is assigned to the SIP protocol under SIP TLS ciphers:



Cipher Settings Page on the Expressway Web Admin Portal



Note: In case of an Expressway cluster, make the changes on the primary server only. The new configuration is replicated to the rest of the cluster members.



Caution: Use the recommended cluster reboot sequence provided in the Cisco Expressway Cluster Creation and Maintenance Deployment Guide. Start by restarting the primary server, wait for it to be accessible via web interface, then do the same with each peer in order according to the list configured under **System > Clustering**.

Disable a Group of Ciphers Using a Common Algorithm

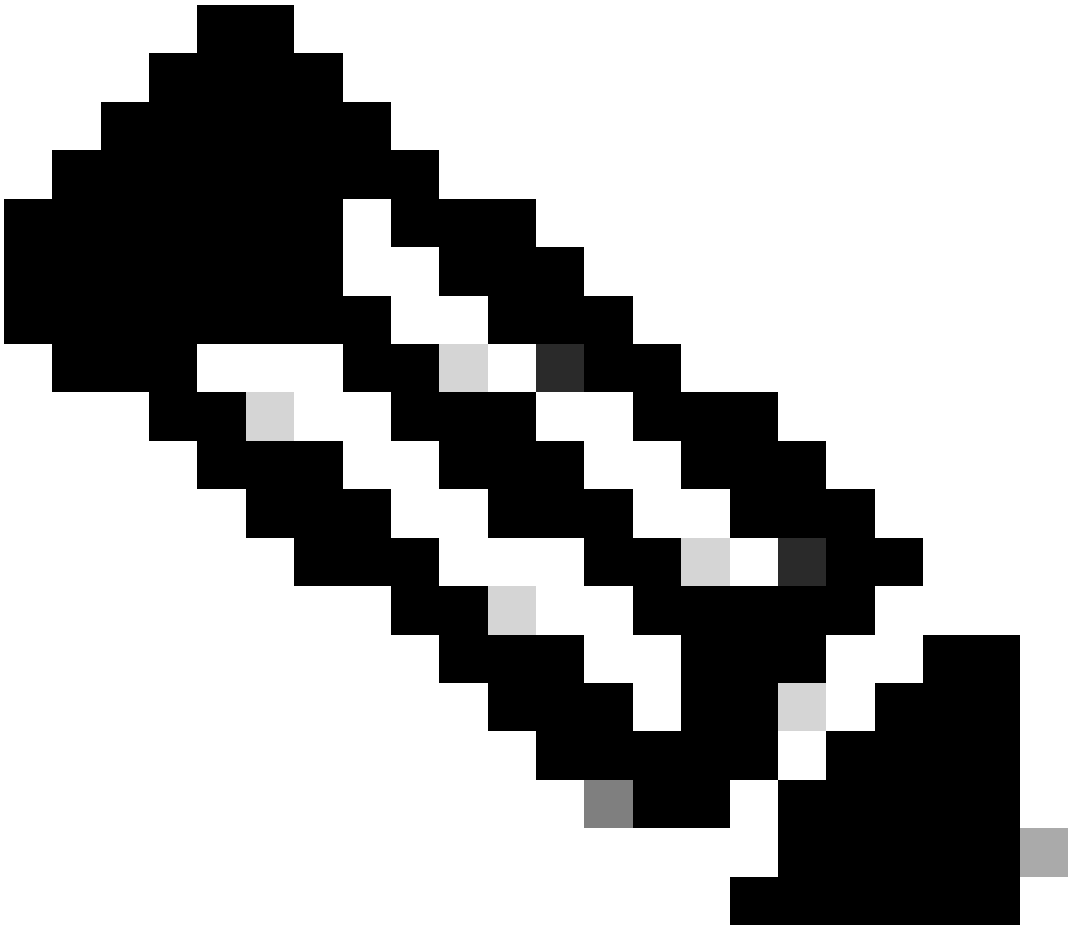
In order to disable a group of ciphers using a common algorithm, append to the default string the **:** separator, the **!** or **-** sign, and the algorithm name to be disabled. The supported algorithm names are available in the OpenSSL Ciphers Manpage. For example, if you need to disable all ciphers that use the DHE algorithm, configure a cipher string like this:

<#root>

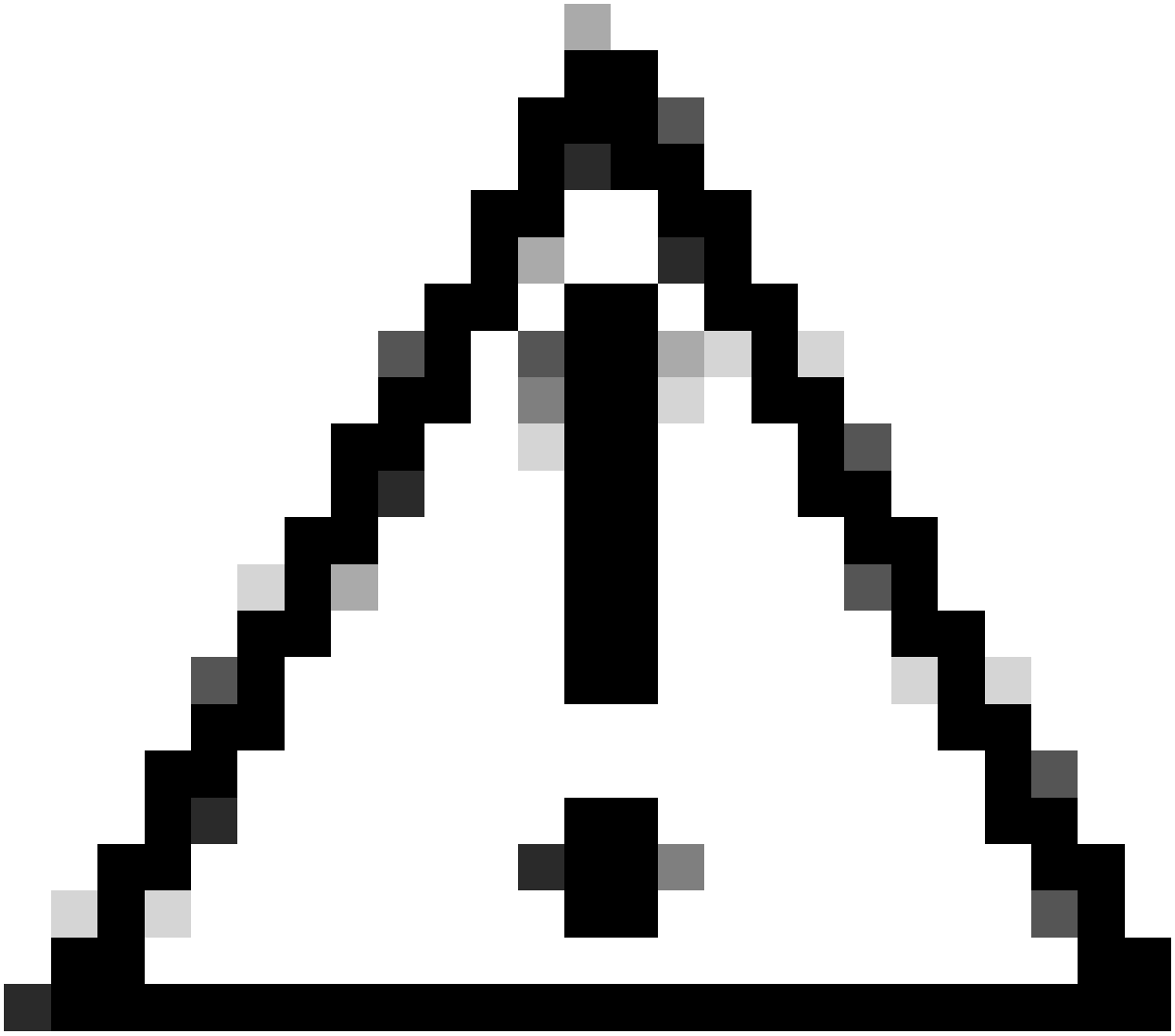
```
EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH
```

:!DHE

Navigate to the **Expressway web admin** page, navigate to **Maintenance > Security > Ciphers**, assign the custom string to the required protocol(s), and click **Save**. For the new configuration to be applied, a system restart is required.



Note: In case of an Expressway cluster, make the changes on the primary server only. The new configuration is replicated to the rest of the cluster members.



Caution: Use the recommended cluster reboot sequence provided in the Cisco Expressway Cluster Creation and Maintenance Deployment Guide. Start by restarting the primary server, wait for it to be accessible via web interface, then do the same with each peer in order according to the list configured under **System > Clustering**.

Verify

Inspect the List of Ciphers Allowed by the Cipher String

You can inspect the customized cipher string by using the **openssl ciphers -V "<cipher string>"** command. Review the output in order to confirm that the undesired ciphers are no longer listed after the changes. In this example, the EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH:!DHE cipher string is inspected. The command output confirms that the string does not allow any of the ciphers that use the DHE algorithm:

<#root>

```
    ~ # openssl ciphers -V "EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH
:!DHE
"
0x13,0x02 - TLS_AES_256_GCM_SHA384 TLSv1.3 Kx=any Au=any Enc=AESGCM(256) Mac=AEAD
0x13,0x03 - TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any Au=any Enc=CHACHA20/POLY1305(256) Mac=AEAD
0x13,0x01 - TLS_AES_128_GCM_SHA256 TLSv1.3 Kx=any Au=any Enc=AESGCM(128) Mac=AEAD
0xC0,0x2C - ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(256) Mac=AEAD
0xC0,0x30 - ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(256) Mac=AEAD
0xCC,0xA9 - ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
0xCC,0xA8 - ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
0xC0,0xAD - ECDHE-ECDSA-AES256-CCM TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESCCM(256) Mac=AEAD
0xC0,0x2B - ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(128) Mac=AEAD
0xC0,0x2F - ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(128) Mac=AEAD
0xC0,0xAC - ECDHE-ECDSA-AES128-CCM TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESCCM(128) Mac=AEAD
0xC0,0x24 - ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA384
0xC0,0x28 - ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA384
0xC0,0x23 - ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA256
0xC0,0x27 - ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA256
0xC0,0x09 - ECDHE-ECDSA-AES128-SHA TLSv1 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA1
0xC0,0x13 - ECDHE-RSA-AES128-SHA TLSv1 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA1
0x00,0x9D - AES256-GCM-SHA384 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(256) Mac=AEAD
0xC0,0x9D - AES256-CCM TLSv1.2 Kx=RSA Au=RSA Enc=AESCCM(256) Mac=AEAD
0x00,0x9C - AES128-GCM-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(128) Mac=AEAD
0xC0,0x9C - AES128-CCM TLSv1.2 Kx=RSA Au=RSA Enc=AESCCM(128) Mac=AEAD
0x00,0x3D - AES256-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA256
0x00,0x3C - AES128-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(128) Mac=SHA256
0x00,0x2F - AES128-SHA SSLv3 Kx=RSA Au=RSA Enc=AES(128) Mac=SHA1
~ #
```

Test a TLS Connection by Negotiating a Disabled Cipher

You can use the **openssl s_client** command in order to verify that a connection attempt using a disabled cipher is rejected. Use the **-connect** option to specify your Expressway address and port, and use the **-cipher** option to specify the single cipher to be negotiated by the client during the TLS handshake:

openssl s_client -connect <address>:<port> -cipher <cipher> -no_tls1_3

In this example, a TLS connection towards Expressway is attempted from a Windows PC with openssl installed. The PC, as the client, negotiates only the undesired DHE-RSA-AES256-CCM cipher, which uses the DHE algorithm:

<#root>

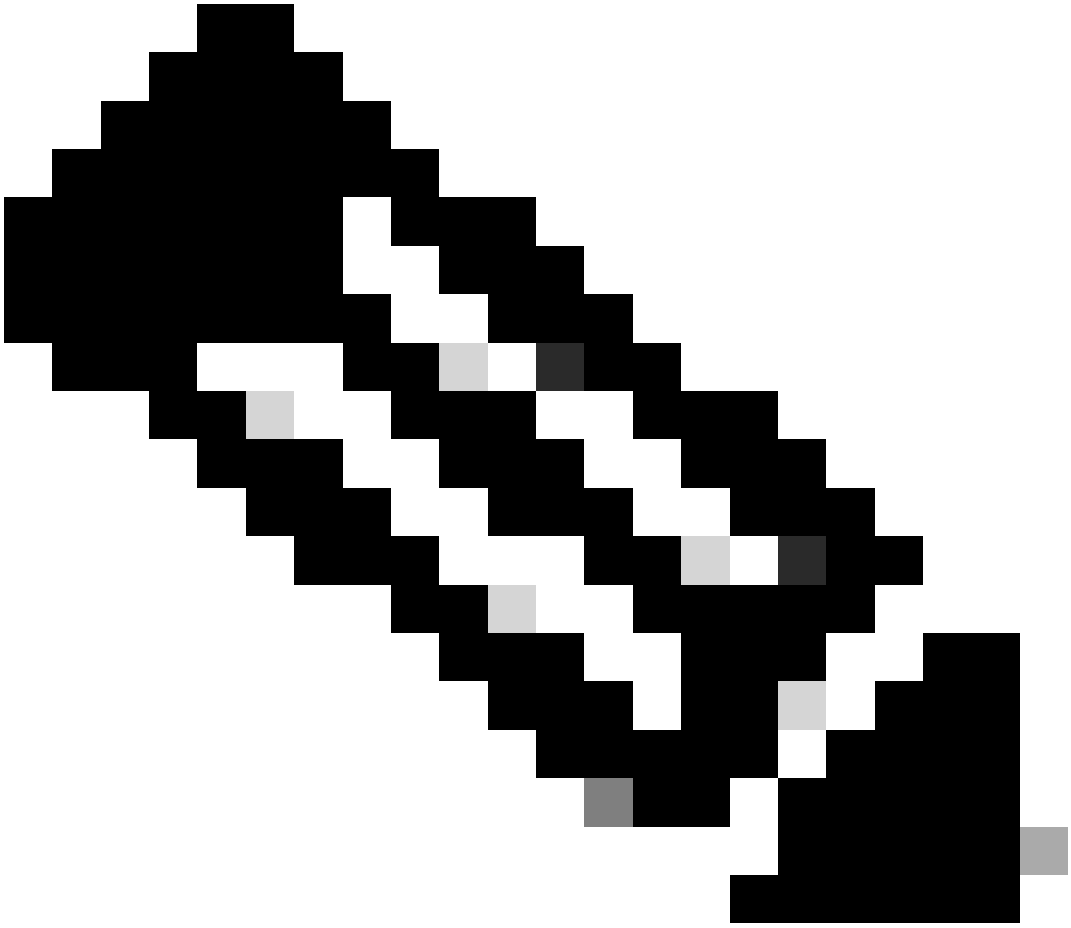
```
C:\Users\Administrator>
openssl s_client -connect exp.example.com:443 -cipher DHE-RSA-AES256-CCM -no_tls1_3


Connecting to 10.15.1.7
CONNECTED(00000154)
D0130000:error:0A000410:SSL routines:ssl3_read_bytes:
ssl/tls alert handshake failure
:..\ssl\record\rec_layer_s3.c:865:
SSL alert number 40
```

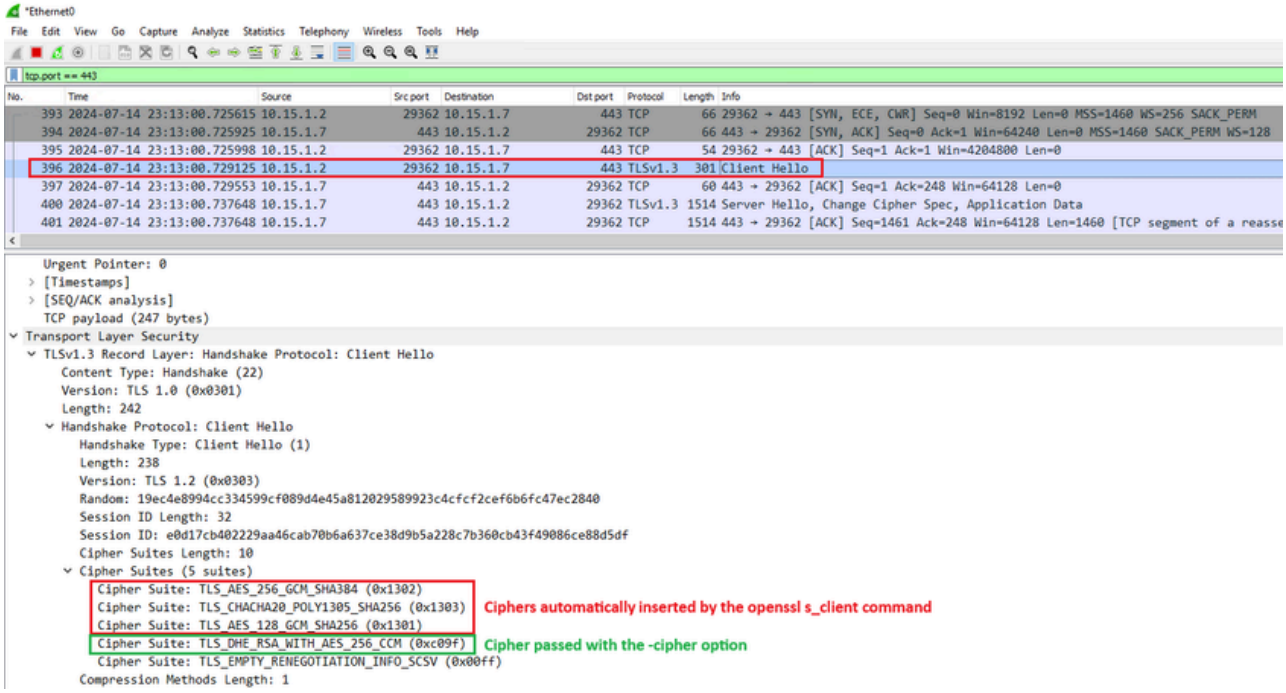
```
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 118 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
No ALPN negotiated
SSL-Session:
Protocol : TLSv1.2
Cipher : 0000
Session-ID:
Session-ID-ctx:
Master-Key:
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1721019437
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
---

C:\Users\Administrator>
```

The command output shows the connection attempt fails with an "ssl/tls alert handshake failure:..\ssl\record\rec_layer_s3.c:865:SSL alert number 40" error message, because the Expressway is configured to use the EECDH:EDH:HIGH:-AES256+SHA:!MEDIUM:!LOW:!3DES:!MD5:!PSK:!eNULL:!aNULL:!aDH:!DHE cipher string for HTTPS connections, which disables ciphers that use the DHE algorithm.



Note: In order for tests with the openssl s_client command to work as explained, the -no_tls1_3 option needs to be passed to the command. If not included, the client automatically inserts TLS 1.3 ciphers in the ClientHello packet:



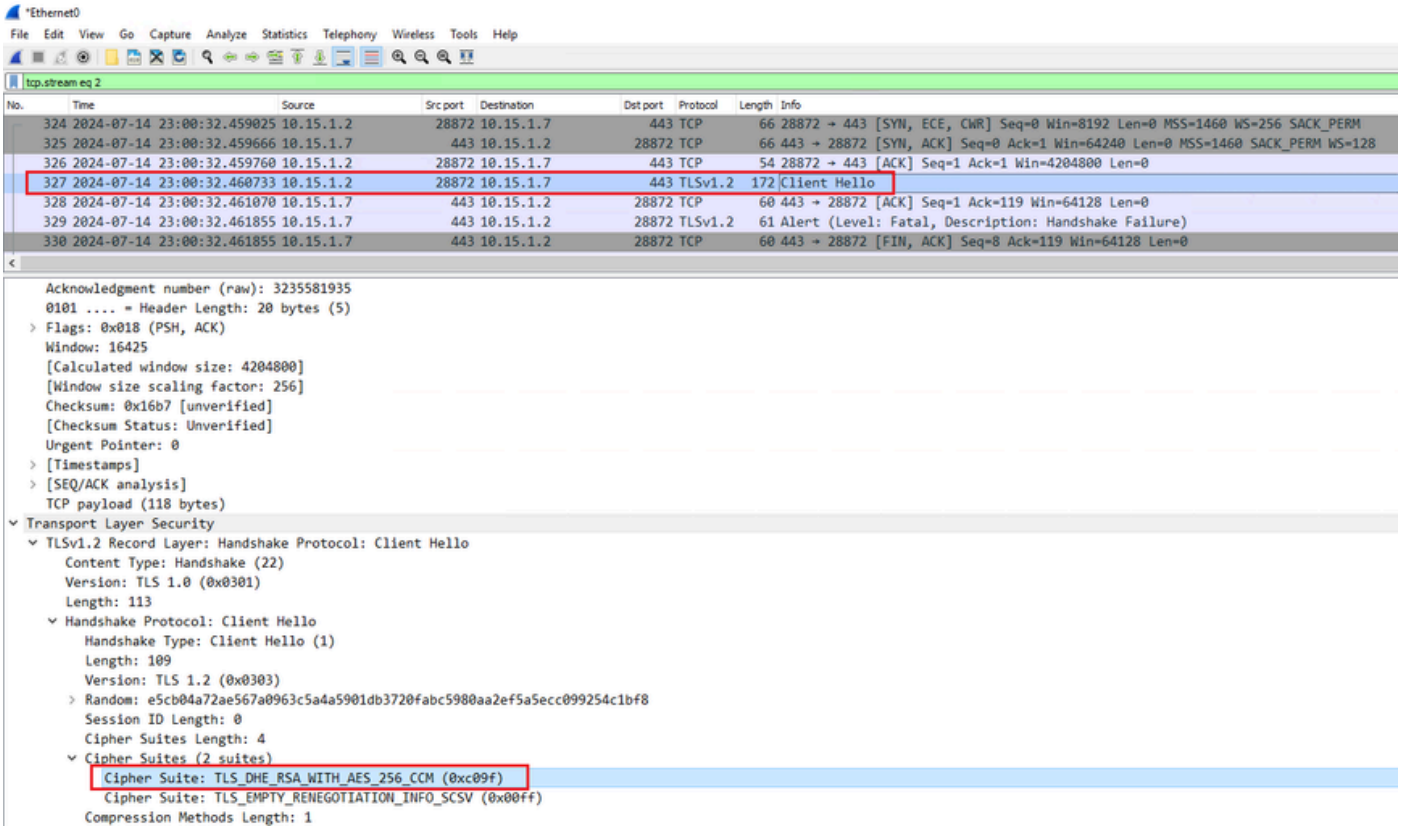
ClientHello Packet With Automatically Added Ciphers

If the target Expressway supports those ciphers, one of them can be chosen instead of the specific cipher that you need to test. The connection is successful, which can lead you to believe that a connection was possible by using the disabled cipher passed to the command with the -cipher option.

Inspect a Packet Capture of a TLS Handshake Using a Disabled Cipher

You can collect a packet capture, from the testing device or from the Expressway, while performing a connection test using one of the disabled ciphers. You can then inspect it with Wireshark in order to further analyze the handshake events.

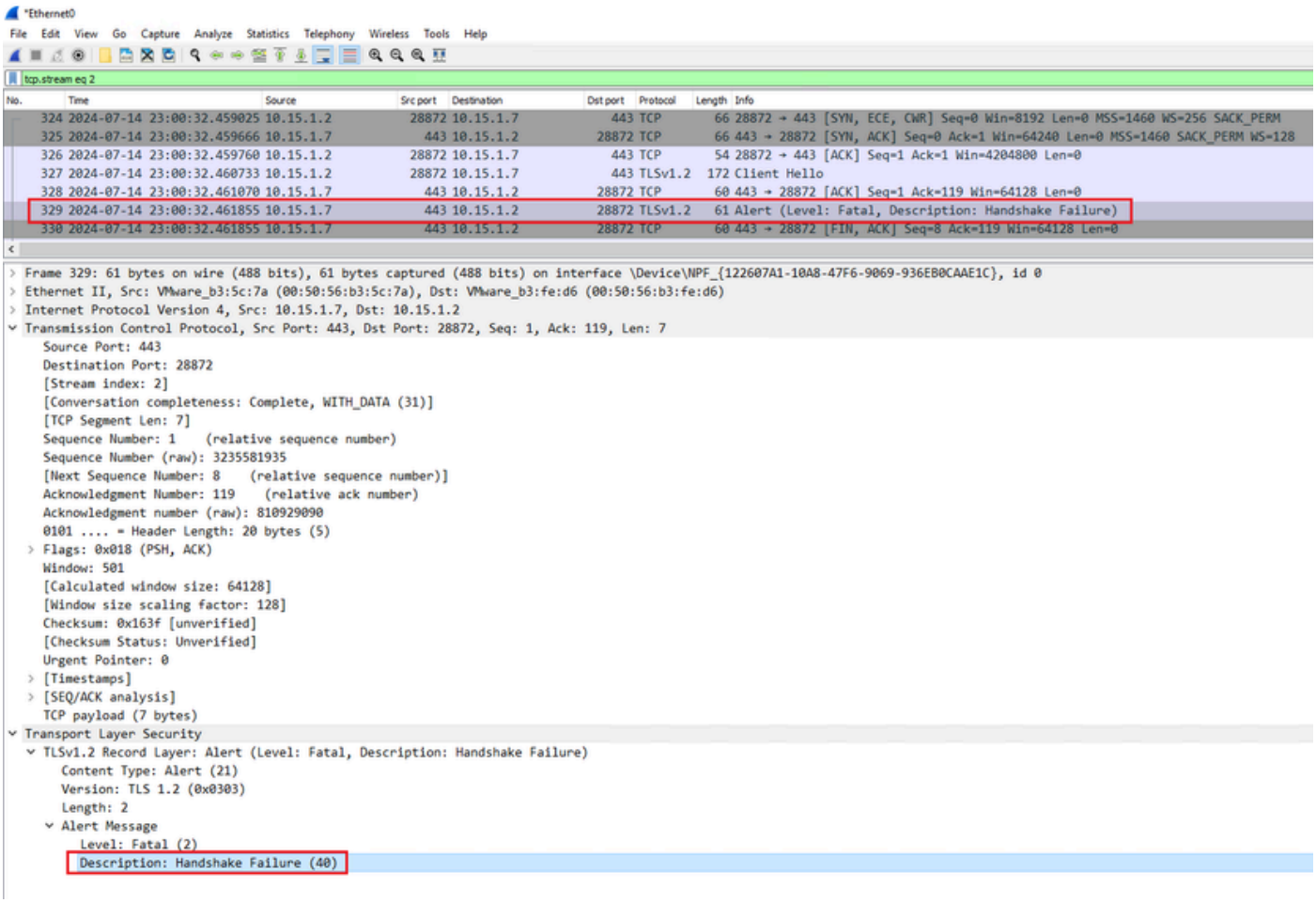
Find the ClientHello sent by the testing device. Confirm that it negotiates only the undesired test cipher, in this example a cipher using the DHE algorithm:



Example of a ClientHello Packet in Wireshark

:

Confirm that Expressway responds with a fatal TLS alert packet, refusing the connection. In this example, since Expressway does not support DHE ciphers per its configured cipher string for the HTTPS protocol, it responds with a fatal TLS alert packet containing failure code 40.



A TLS Fatal Alert Packet in Wireshark

Related Information

- OpenSSL Ciphers Manpage
- Cisco Expressway Administrator Guide (X15.0) - Chapter: Managing Security - Configuring Minimum TLS Version and Cipher Suites