TLS Configuration Guide for ONOS Controller Interfaces

This repository hosts the documentation of the security implementation process of the various interfaces of the ONOS controller, for a dissertation within the scope of the Master's in informatics Security, whose purpose is to develop in-depth knowledge of the SDN network, its protocols, its architecture and concept, having as final objective the development and validation of a security framework for SDN networks.

Context

Software-defined networks (SDN) changed the traditional paradigm of traditional networks, bringing a programmable, scalable network with greater management capacity. The SDN network paradigm separates the control plane from the data plane, which results in the ease of configuration and management of changes in a network, in this way the controller is responsible for routing decisions when they cannot be handled by others. components, it also allows the implementation of better and more consistent control policies in a uniform way for the entire network through the controller.

With the introduction of new elements in the network and the change in architecture, the security of SDN networks has become a point of concern as it has brought new challenges and vulnerabilities that can lead to more serious consequences for the network. Among the most relevant challenges is the problem of centralized controllers that introduce single points of failure (Single Point of Failures), being vulnerable to distributed denial of service (DDos) attacks. There are also problems with data privacy, authentication, integrity, and access of data on the network by third parties. In this sense, in order to overcome these security challenges, an objective choice of the controller proves to be a very important point, where its operation in cluster mode must be considered in order to overcome the single points of failure, the information of the vulnerabilities that it presents, its functionalities and security mechanisms.

Architecture

The security framework is modeled on a distributed architecture consisting of three controllers inserted in a cluster, whose objective is to use the mechanisms existing in the controller itself to implement security mechanisms for all interfaces in order to protect communications, specifically the implementation of the TLS protocol on all the interfaces

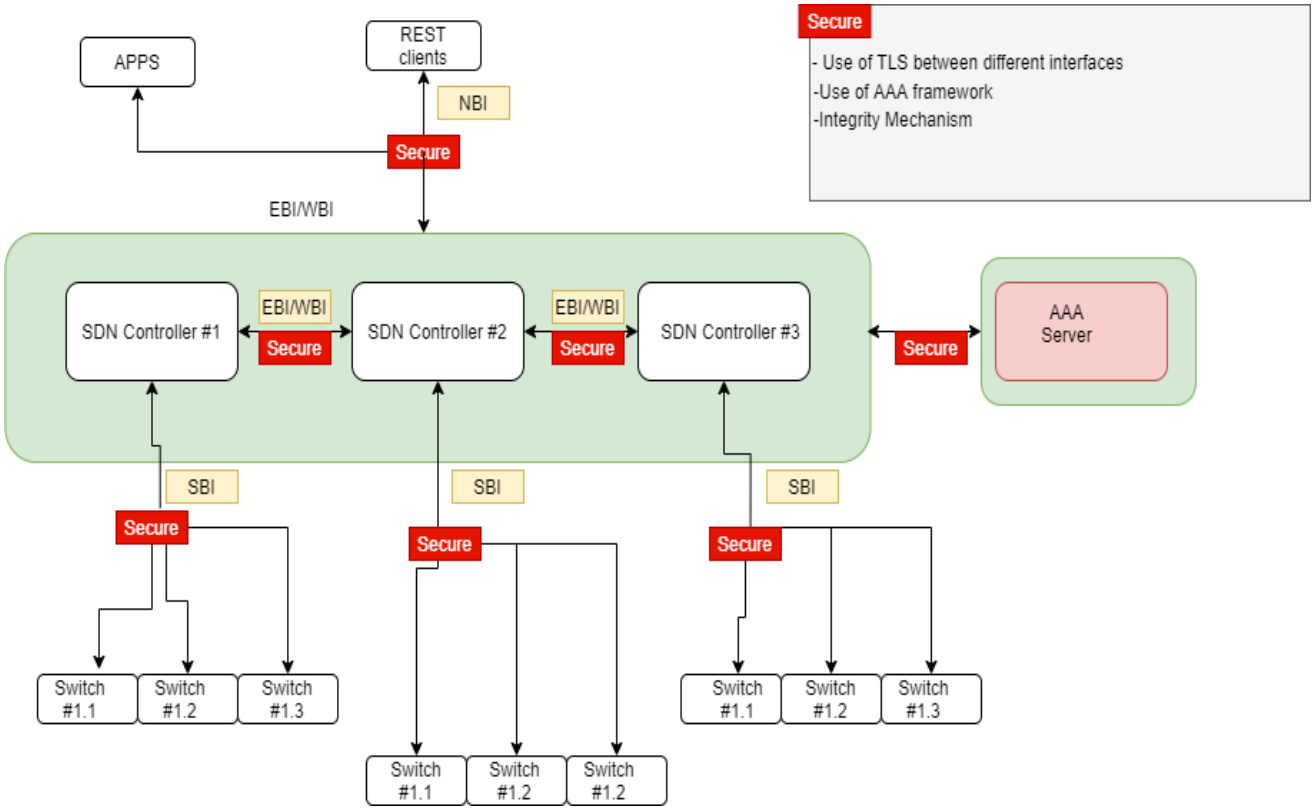


Figure 1- *Interface security implementation scenario (NBI, SBI, EBI and WBI).*

Pre-requisites

It is recommended that the prototype be implemented in Linux environments, because no tests were performed for MacOS and Windows operating systems, which can lead to unpredictable dependencies and behavior in these operating systems.

What you'll need to install:

- Docker Engine – Community
    - Version: 20.10.7

- Atomix
    - atomix:3.1.5
- ONOS
    - onosproject/onos:2.2.2

Environment

The tables below represent the IP addresses used in the scenario.

Host Name	IP Address	OS
Node-1	192.168.5.145	CentOS 7
Node-2	192.168.5.146	CentOS 7
Node-3	192.168.5.150	CentOS 7

Atomix nodes

Host Name	IP Address	version
atomix1	172.20.0.10	atomix:3.1.5
atomix2	172.20.0.11	atomix:3.1.5
atomix3	172.20.0.12	atomix:3.1.5

ONOS nodes

Host Name	IP Address	version
onos1	172.20.0.20	onos:2.2.2
onos2	172.20.0.21	onos:2.2.2
onos3	172.20.0.22	onos:2.2.2

Configure Atomix Files

The cluster creation process requires a configuration of two different files in each of the cluster nodes. The first is the Atomix configuration file in JSON format which requires three essential components:

Cluster

- Specifies how the Atomix nodes communicate and discover each other's in the cluster.

Partition Group

- Configure membership groups for storage and replication.

Management Group

- Configure partition groups to manage primitives.

Tabela 1-Atomix.conf

Atomix.conf

```
1:{
2:   "cluster": {
3:       "node": {
 4:         "id": "atomix-1",
 5:         "address": "YourAtomixNodeIP"
 6:      },
 7:     "clusterId": "onos",
 8:      "discovery": {
 9:         "nodes": [
 10:             {
 11:               "id": "atomix-1",
 12:               "address": " YourAtomixNodeIP "
 13:             },
 14:            {
 15:               "id": "atomix-2",
 16:               "address": " YourAtomix2NodeIP "
 17:             },
18:
 19:             {
 20:                "id": "atomix-3",
 21:               "address": " YourAtomix3NodeIP "
 22:            }
23:
 24:          ],
 25:          "type": "bootstrap"
 26:      }
27:  },
28:   "partitionGroups": {
 29:      "raft": {
 30::         "partitionSize": 3,
 31:         "storage": {
 32:           "level": "mapped"
 33:         },
 34:          "type": "raft",
 35:         "members": [
 36:             "atomix-1",
 37:             "atomix-2",
 38:             "atomix-3"
 39:          ],
 40:          "partitions": 3
 41:      }
 42:  },
 43:  "managementGroup": {
 44:     "partitionSize": 2,
 45:      "storage": {
 46:       "level": "mapped"
 47:     },
 48:     "type": "raft",
 49:     "members": [
```

```
50:       "atomix-1",
 51:       "atomix-2",
  52:       "atomix-3"
  53:    ],
   54:   "partitions": 1
  55: }
```

In the cluster configuration the nodes provided for the discovery protocol are the set of all Atomix nodes (line 8 to 21). Note that ONOS nodes don't need to be mentioned, as ONOS is connected to Atomix in a client-server type architecture. The address parameter for each node is represented by host:port. The bootstrap protocol is recommended for consensus nodes (line 25), as fixed node identities are a requirement of the Raft protocol.

In the configuration of partition groups (line 28 to 40) a set of named groups is defined to store distributed primitives. Note that ONOS expects a single Raft partition group to be configured. The group can be named with any name and can include any number of partitions as long as they are set to 1 or 2 times the number of Atomix nodes

The management group (line 43 to 54) defines the protocol and configuration to be used to administer the Atomix cluster. As ONOS relies on Atomix for consensus, this should be a Raft partition group. The management group can be configured with just a single partition as the load on the group is low.

Configure ONOS Files

It is necessary to configure ONOS to be able to locate and connect to the Atomix nodes, for that the cluster.json file is configured on each cluster host.

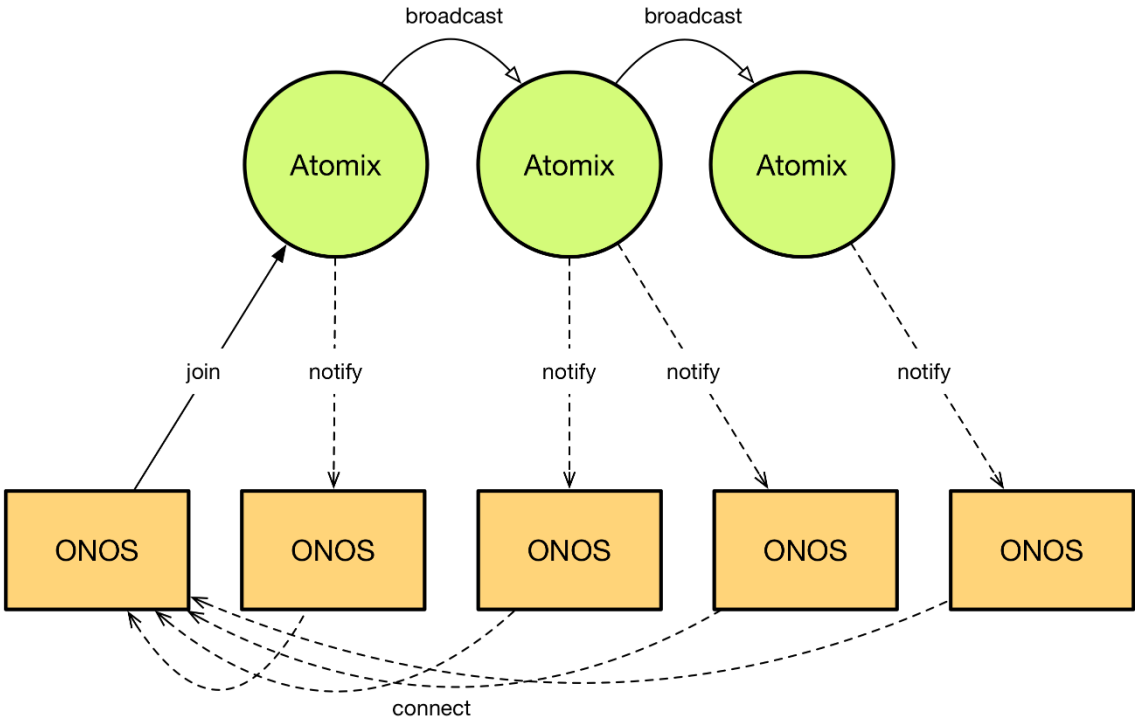
Tabela 2-Cluster.json

Cluster.json

```
1:{
2:    "node": {
3:      "ip": "ONOSNodeIP",
4:       "id": " ONOSNodeIP ",
5:       "port": 9876
6:    },
7:   "storage": [
8:       {
9:          "ip": " YourAtomix1NodeIP ",
10:          "id": "atomix-1",
11:          "port": 5679
12:       },
 13:      {
14:          "ip": " YourAtomix2NodeIP ",
 15:         "id": "atomix-2",
 16:         "port": 5679
 17:      },
 18:      {
```

```
19:        "ip": " YourAtomix3NodeIP ",
20:         "id": "atomix-3",
21:         "port": 5679
22:      }
23:   ],
24:  "name": "onos"
25:      }
```

When an ONOS node is initialized, it connects to the external Atomix cluster for data storage and coordination. When connecting, the ONOS node notifies Atomix of its existence and location, which in turn transmits the information from the new node to all other connected ONOS nodes, and the connected ONOS nodes consequently connect directly back to the new ONOS node for point-to-point communication.



Cluster configuration

With the Atomix.conf and cluster.json files created, the next step is to add the clustered hosts, for that it is necessary to use the Docker swarm tool. The first host will be the Swarm master.

CREATE DOCKER SWARM

1- To start the cluster on the master node we use the command:

Tabela 3-Docker swarm

MASTER
docker swarm init

2- The output of this command will generate a unique token needed to add other nodes to the cluster, for that you must copy and paste the command into the “slave” nodes. The command below exemplifies the execution of this command:

Tabela 4-docker swarm join

Slaves
docker swarm join --token SWMTKN-1-3tqtr2s7iishum1f2j-7e91x9ccw25upu2u1e9t6vdl3 192.168.5.145:2377

3- To verify that the commands were executed correctly and that the swarm was created, it is necessary to execute the following command on the manager node. The command output presents a list of all nodes.

Tabela 5-docker node ls

MASTER
docker node ls

CREATE DOCKER NETWORK

4- With the hosts already added to the cluster, it is necessary to create the network for communication between them. For this, an overlay network is created that will work as a virtual network on top of the existing network. This command is advised to be executed on the master or manager node.

Tabela 3-docker network create

MASTER
docker network create --driver=overlay --attachable <name of the network> --subnet=<subnet range>
docker network create -d overlay --attachable onos --subnet 172.20.0.0/16 --gateway 172.20.0.1

5- To verify the created network, we can run the command below.

Tabela 4-docker network ls

MASTER
docker network ls

CREATE ATOMIX DOCKER

6- With the hosts clustered and the network needed for communication created, the next step is to create the atomix nodes instances. This command must be executed in each of the nodes, once a docker with the desired name is created, it is assigned an ip address of the docker network created previously. Added the "--restart unless-stopped" argument, so containers would restart if it were stopped or even after a system restart.

Tabela 5-docker create atomix

Nodes	
Node1	docker create -t -p 5679:5679 --restart unless-stopped --name atomix-1 --hostname atomix-1 --net onos --ip 172.20.0.10 atomix/atomix:3.1.5
Node2	docker create -t -p 5679:5679 --restart unless-stopped --name atomix-2 --hostname atomix-2 --net onos --ip 172.20.0.11 atomix/atomix:3.1.5
Node3	docker create -t -p 5679:5679 --restart unless-stopped --name atomix-3 --hostname atomix-3 --net onos --ip 172.20.0.12 atomix/atomix:3.1.5

7- You must add environment variables to the IP of the atomix instances obtained in the command above

Tabela 6-docker variables IP

Nodos	
Node1	export OC1=172.20.0.10
Node2	export OC2=172.20.0.11
Node3	export OC3=172.20.0.12

8- The Atomix.conf files generated in the first step must be copied to the atomix instances created in step 6.

Tabela 7-Copy Atomix configuration to docker instances

Nodos	**Instâncias Atomix**
Node1	docker cp ~/atomix1.conf atomix- 1:/opt/atomix/conf/atomix.conf
Node2	docker cp ~/atomix2.conf atomix-2:/opt/atomix/conf/atomix.conf
Node3	docker cp ~/atomix3.conf atomix-3:/opt/atomix/conf/atomix.conf

9- Finally, you must initialize the atomix instances for the settings to be applied

Tabela 8- Restart Atomix docker instances

Nodos	**Instâncias Atomix**
Node1	docker container start atomix-1
Node2	docker container start atomix-2
Node3	docker container start atomix-3

CREATE ONOS DOCKER

10- To create the ONOS instances the process is like creating the atomix, with the difference in the arguments passed in the command. When executed, the onos instances are created where it is assigned an ip address previously referenced in the creation of the cluster.json configuration file. The argument “ONOS_APPS” was added, this way the containers would be restarted with the chosen applications.

Nodos	**Instâncias ONOS**
Nodo1	docker run -t -d -p 80:8181 -p 8443:8443 -p 6653:6653 -p 6633:6633 -p 5005:5005 -p 830:830 --restart unless-stopped --name onos1 --hostname onos1 --net onos --ip 172.20.0.20 -e ONOS_APPS="drivers,openflow,lldpprovider,proxyarp,fwd,gui2" onosproject/onos:2.2.2
Nodo2	docker run -t -d -p 80:8181 -p 8443:8443 -p 6653:6653 -p 6633:6633 -p 5005:5005 -p 830:830 --restart unless-stopped --name onos2 --hostname onos2 --net onos --ip 172.20.0.21 -e ONOS_APPS="drivers,openflow,lldpprovider,proxyarp,fwd,gui2" onosproject/onos:2.2.2
Nodo3	docker run -t -d -p 80:8181 -p 8443:8443 -p 6653:6653 -p 6633:6633 -p 5005:5005 -p 830:830 --restart unless-stopped --name onos3 --hostname onos3 --net onos --ip 172.20.0.22 -e ONOS_APPS="drivers,openflow,lldpprovider,proxyarp,fwd,gui2" onosproject/onos:2.2.2

Tabela 9-Create docker instances

11- The next step is to create the configuration directory for the ONOS docker instances where the cluster.json files will be copied.

Nodos	**Instâncias ONOS**
Node1	docker exec onos1 mkdir /root/onos/config
Node2	docker exec onos1 mkdir /root/onos/config
Node3	docker exec onos1 mkdir /root/onos/config

Tabela 10-Create config directory for ONOS instances

12- With the created directories it is necessary to copy the files to the ONOS instances.

Nodos	**Instâncias ONOS**
Node1	docker cp ~/cluster-1.json onos1:/root/onos/config/cluster.json
Node2	docker cp ~/cluster-2.json onos2:/root/onos/config/cluster.json
Node3	docker cp ~/cluster-3.json onos3:/root/onos/config/cluster.json

Tabela 11-Copy ONOS cluster configuration to docker instance

13- The ONOS instances must be rebooted for the settings to be applied.

Nodos	**Instâncias Atomix**
Nodo1	docker restart onos1
Nodo2	docker restart onos2
Nodo3	docker restart onos3

TLS CONFIGURATION

With the cluster created, the next step was to identify the mechanisms that already exist in the controller that allows protecting its communication interfaces. The TLS protocol uses version 1.2 where the RSA algorithm was configured as it is one of the most secure algorithms and is a good option for encrypting messages transmitted through TLS. The basis of RSA security is the level of difficulty in factoring very large numbers, where the larger the key size the higher the security level, so the recommended size implemented was a 2048 bit size key which is also the base provided by a certification authority.

Generate a Key/Certificate

1-The command below will generate a new key and add it to the keystore. The 'alias', 'storepass', 'validity' and 'key size' values can be modified as per the required requirements. The 'keyalg' value is recommended to use RSA as the default DSA value can cause errors in some browsers.

Tabela 12-Generate a certificate

Generate a Key/Certificate
keytool -genkey -alias teastoressl -keyalg RSA -keysize 2048 -validity 360 -keystore "/rootapache-karaf-4.2.8/etc/keystore/rc.keystore" -ext SAN=dns: YourONOSInstanceName, ip:HostIP What is your first and last name? [CV]: **InstanceName** What is the name of your organizational unit? [CV]: **UnitName** What is the name of your organization? [CV]: **OrganizationName** What is the name of your City or Locality? [CV]: **CityName** What is the name of your State or Province? [CV]: **StateName** What is the two-letter country code for this unit? [CV]: **CountryName** Is CN=**InstanceName**, OU=**dei**, O=**uc**, L=**coimbra**, ST=**coimbra**, C=**pt** correct? [no]: yes, Enter key password for (RETURN if same as keystore password):

Certificate Distribution

2-With the certificate generated in each ONOS node, it is necessary to distribute it among other ONOS nodes. For this, the key must be converted into a file. p12 and then in a PEM file, it can be run with the following commands:

Tabela 15-Get the certificate

Get the certificate
keytool -importkeystore -srckeystore /root/onos/apache-karaf-4.2.8/etc/keystore/rc.keystore -destkeystore onos.p12 -srcstoretype jks -deststoretype pkcs12
openssl pkcs12 -in onos.p12 -out onos.pem

The certificate can be extracted from the PEM file using the command above. The output file of this command must be adjusted to ensure that it is unique for each node of the ONOS.

Tabela 16-Extract PEM file

Extract PEM file
awk 'split_after == 1 {n++; split_after=0} /-----END ENCRYPTED PRIVATE KEY-----/ {split_after=1} {print > "onoscert" n ". pem"}' < onos1.pem; mv onoscert1.pem cert.pemd
mv cert.pem onos-<hostname>-cert.pem

Copy and Import certificates for each ONOS node

3-For each ONOS instance present in the cluster it will need the certificate for the remaining instances. The Certificates can be copied using the following command and must be repeated for each instance and each certificate, with the certificates copied to each instance it will need to have them in its keystore.

Tabela 17-Copy and Import certificates

Copy certificates from ONOS container to host
docker cp “container ID” /root/onos/apache-karaf-4.2.8/etc/onos1cert.pemd /home/node
Share certificate for other ONOS nodes
scp /home/node-3/onos3.pem node-1@192.168.5.145:/home/node-1/ scp /home/node-3/onos3.pem node-1@192.168.5.146:/home/node-2/
Send the received certificates to the ONOS keystore
docker cp /home/node-2/onos1.pem onos3:/root/onos/apache-karaf-4.2.8/etc/keystore docker cp /home/node-2/onos3.pem onos3:/root/onos/apache-karaf-4.2.8/etc/keystore
Convert certificates to format. pem to .crt to be added to the Keystore
openssl x509 -outform der -in onos1.pem -out onos1cert.crt openssl x509 -outform der -in onos3.pem -out onos2cert.crt
Import converted certificates to keystore
keytool -import -alias onos3 -file onos2cert.crt -keystore /root/onos/apache-karaf-4.2.8/etc/keystore/rc.keystore
Check the certificates imported in the keystore of each ONOS node
keytool -v -list -keystore /root/onos/apache-karaf-4.2.8/etc/keystore/rc.keystore

TLS on the Northbound Interface

The access to Northbound interface is done through WEB UI and REST API provided by HTTP protocol. By default, user access is required authentication, but transfer between client and server is not, which can allow an attacker to change and observe data exchanged between client and server, so it is necessary to configure and enable the HTTPS protocol to ensure secure communication with the Northbound interface.

Modify Configuration

1-With the created key, it is necessary to change the “org.ops4j.pax.web.cfg” file from ONOS according to the command below, where the passwords and keystore parameters must be defined according to what was configured in the step of generating certificate and key.

Tabela 18-Change the onos file

org.ops4j.pax.web.cfg
1: org.osgi.service.http.port=**8181** 2: org.osgi.service.http.port.secure=**8443** 3: 4: org.osgi.service.http.enabled=**true** 5: org.osgi.service.http.secure.enabled=**true** 6: 7: org.ops4j.pax.web.ssl.keystore=etc/keystore 8: org.ops4j.pax.web.ssl.password=**onos22** 9: org.ops4j.pax.web.ssl.keypassword=**onos22** 10: 11: 12: org.ops4j.pax.web.session.timeout=1440 13: 14 : org.ops4j.pax.web.session.url=none 15: org.ops4j.pax.web.config.file=./etc/jetty.xml

Connecting Over HTTPS

2-Some browsers do not trust the generated certificate and will need to add a security exception. This can be done using a certification authority to sign the certificate. Another solution that can be used is to add the certificate directly to the system's trusted certificates. On windows system it can be add in the Microsoft Management Console. With the generated certificates, the ONOS web UI can be accessed through:

https://localhost:8443/onos/ui

TLS on the East/West Interface

Communication between controllers is done on the East/West interface which by default does not provide secure communication, which allows the observation and change of traffic between controllers. TLS must be enabled for between controllers to ensure communication within the ONOS cluster is secure. The TLS configuration process can be carried out bearing in mind the information on the ONOS website, it should be noted that some parameters and other steps were added due to changes in the most recent versions of ONOS and in the formation of the cluster through Atomix, specifically it was added the parameter "*cluster.messaging.tls*" in the configuration file of the Atomix nodes, changes that the ONOS website does not present as it had its last update date in the year 2018. Before proceeding, it is necessary to make the previous steps of Distribution of certificates and import of certificates for each node of Atomix.

Modify atomix.conf File

1-In the most recent versions of ONOS the formation of the cluster has changed where we have the separation of Atomix and ONOS, which was previously embedded within ONOS. In this way, it is necessary to change the Atomix.conf file in each Atomix node of the cluster, where a new parameter "*cluster.messaging.tls*" is added.

Tabela 19-Change the atomix.conf file

Atomix.conf
1: cluster.messaging.tls: 2: 3: { 4: enabled: true 5: 6: keyStore: /opt/atomix/onos/apache-karaf-4.2.8/etc/keystore/rc.keystore 7: 8: keyStorePassword: onos22 9: 10: trustStore: /opt/atomix/onos/apache-karaf-4.2.8/etc/keystore/rc.keystore 11: 12: trustStorePassword: onos22 13: 14: }

Change file in ONOS

2-For each instance of ONOS it is necessary to change the script of the “onos-service” file located in the ONOS_ROOT / bin / directory, where it contains a line to activate the TLS protocol for Netty. Uncomment the line and make sure that the values of the 'keystore' and 'truststore' parameters point to the keystore used in the commands described above.

Tabela 20-Change onos-service file

Change onos-service file
Export JAVA_OPTS="${JAVA_OPTS: --DenableNettyTLS=true -Djavax.net.ssl.keyStore=/root/onos/apache-karaf-4.2.8/etc/keystore/keystore/rc.keystore -Djavax.net.ssl.keyStorePassword=onos22 -Djavax.net.ssl.trustStore=/root/onos/apache-karaf-4.2.8/etc/keystore/keystore/rc.keystore -Djavax.net.ssl.trustStorePassword=onos22}"

TLS on the Southbound Interface

The Southbound interface does not encrypt data or authenticate devices connected to the network. With TLS enabled, communication between ONOS and devices in the data plane (example: routers, switchers) are encrypted, ensuring privacy in communications, and alteration of data traffic.

Convert Files

With the certificate generated previously , it will be used by the OvS (OpenvSwitch), which supports files in .pem format, so it must be converted through a two-step process. In the first one, the keystore must be converted to .p12 format:

Tabela 13-Get certificate in .p12 format

Get certificate in .p12 format
1: keytool -importkeystore -srckeystore apache-karaf-3.0.8/etc/keystore/rc.keystore --destkeystore onos1.p12 -srcstoretype jks -deststoretype pkcs12: Enter destination keystore password: Re-enter new password: Enter source keystore password: Entry for alias onos successfully imported. Import command completed: 1 entries successfully imported, 0 entries failed or cancelled

With the file in .p12 format it can now be converted to a file in format. Pem

Tabela 14-Export p12 certificates to pem

Exportar ficheiro do formato p12 para pem
1:openssl pkcs12 -in onos1.p12 -out onos1.pem Enter Import Password: MAC verified OK Enter PEM pass phrase: Enter PEM pass phrase

The certificate contained in the 'onos1.pem' file should be copied to a separate file, which will later be shared between the hosts.

Tabela 15-Get certificate to be shared

Exportar ficheiro do formato p12 para pem
1: awk 'split_after == 1 {n++;split_after=0} /-----END ENCRYPTED PRIVATE KEY-----/ {split_after=1} {print > "cacert" n ".pem"}' < onos1.pem; mv cacert1.pem onos1-pub.pem

Copy certificate to OpenvSwitch board

The nos1-pub.pem file must be copied to the appropriate OpenvSwitch (OvS) directory so that it can accept the certificate in the ONOS nodes.

Tabela 16-Copy certificate to OVS

Copy certificate to OVS
1: scp ./ onos1-pub.pem admin@ovs :/var/lib/openvswitch/pki/controllerca

Generate SSL certificate for OpenvSwitch

The host responsible for the OvS components will generate a certificate that will be distributed and used by ONOS, and finally the OvS is configured to use the keys generated in Table 29. The responsible host host corresponds to the node where the network is emulated using the mininet.

Tabela 17-Generate certificate for OvS

Generate certificate for OvS
1: ovs-pki req+sign sc switch

The command below is required on the virtual host in terms of certificates, private key and ONOS controller certificate to encrypt communications. These settings are used by all OvS elements.

Tabela 18-OVS to use as new keys

OVS to use as new keys
1: ovs-vsctl set-ssl /etc/openvswitch/sc-privkey.pem \ /etc/openvswitch/sc-cert.pem /var/lib/openvswitch/pki/controllerca/ onos1-pub.pem

Copy key to ONOS

You need to copy the OvS certificate to the ONOS directory for it to be added to your keystore. The commands in Table 31 represent copying the sc-cert.pem key (generated in Table 29) and importing it into the ONOS host keystore.

Tabela 19-Copy OVS key to ONOS

Copy OVS key to ONOS
1: scp ./sc-cert.pem node-1@192.168.5.145:/ home/node-1/
Import OvS key to ONOS keystore
keytool -importcert -file sc-cert.pem -keystore /root/onos/apache-karaf-4.2.8/etc/keystore/rc.keystore

Change file in ONOS

With the certificate added in the ONOS keystore (Table 31) it is necessary to add new parameters in the "onos-service" file in the ONOS_ROOT / bin / directory in order to activate TLS between the ONOS and the OvS switches.

Change onos-service file
export JAVA_OPTS="${JAVA_OPTS:--DenableOFTLS=true -Djavax.net.ssl.keyStore=/root/onos/apache-karaf-4.2.8/etc/rc.keystore -Djavax.net.ssl.keyStorePassword=onos22 -Djavax.net.ssl.trustStore=/root/onos/apache-karaf-4.2.8/etc/rc.keystore -Djavax.net.ssl.trustStorePassword=onos22}"