**Подключение внешних ключа и сертификата в Istio CA**

В инструкции описан процесс конфигурирования mTLS в Istio с использованием внешних сертификатов (например предоставленных администратором), для подписи сертификатов приложений в Istio. В данном случае ключ и сертификат CA генерируются самостоятельно.

Текущий вариант конфигурирования mTLS в Istio является промежуточным. В целевом варианте планируется интеграция с HSM для получения сертификатов приложений Istio.

Для включения в кластере Istio mTLS, при котором управлением сертификатами и ключами приложений занимается CA Istio, необходимо:

1. Подготовить ключи и сертификаты CA, которые будут использоваться Цитаделью для подписи сертификатов приложений

2. Сконфигурировать Istio CA, на использование внешнего ключа и сертификата CA

3. Настроить Policy в кластере, что бы был разрешен только TLS трафик.

**1. Подготавка ключей и сертификатов.**

Для подготовки self-sign ssl сертификата с CA требуется openssl.

Что бы максимально съэмулировать ситуацию с получением ключа и сертификата подписанного УЦ Банка, нам потребуется сгенерировать корневой сертификат и сертификат для Istio CA, подписанный корневым.

Перед генерацией рекомендуется перейти в корневую директорию с исходными файлами дистрибутива Istio (если применимо). Все сгенерированные ключи и сертификаты будут находиться в директории “./custom/certs/”

Создаем ключь и сертификат Root CA

openssl genrsa -des3 -passout pass:123456 -out custom/certs/rootCA.key 2048

openssl req -x509 -new -nodes -key custom/certs/rootCA.key -days 3650 -out custom/certs/root-cert.pem

Создаем ключь и сертификат для Istio CA (Цитадель не запускается с заданным pass ключа ca-cert)

openssl genrsa -out custom/certs/ca-key.pem 2048

openssl req -new -key custom/certs/ca-key.pem -out custom/certs/ca-cert.csr -subj "/C=RU/ST=MSK/L=MSK/O=Istio test/OU=IT/CN=Istio CA"

openssl x509 -req -extfile <(printf "basicConstraints=critical,CA:TRUE \nkeyUsage=keyCertSign") -in custom/certs/ca-cert.csr -CA custom/certs/root-cert.pem -CAkey custom/certs/rootCA.key -CAcreateserial -out custom/certs/ca-cert.pem -days 3650

Так же нам потребуется файл cert-chain для установки цепочки доверия, т.к. наш сертификат Istio CA (ca-cert.pem) отличается от корневого (root-cert.pem)

Между сертификатом Istio CA и корневым сертификатом, больше нет промежуточных CA, поэтому наш cert-chain.pem должен соедиржать данные только ca-cert.pem

cat custom/certs/ca-cert.pem > custom/certs/cert-chain.pem

**2. Конфигурирование Istio CA**

Офф.документация - <https://archive.istio.io/v1.4/docs/tasks/security/plugin-ca-cert/>

ISTIO\_HOME=$HOME/istio-1.4.0

Создаем секрет на основе ранее сгенерированных ключей и сертификатов, для Цитадели

oc create secret generic cacerts -n istio-system --from-file=custom/certs/ca-cert.pem --from-file=custom/certs/ca-key.pem --from-file=custom/certs/root-cert.pem --from-file=custom/certs/cert-chain.pem

Преконфигурируем Цитадель.

В рамках конфигурирования:

* Устанавляиваем mtls для межсервисных взаимодействий в кластере в качестве значения по умолчанию - настройка global.mtls.enabled=true
* Отключаем генерацию Цитаделью собственных сертификатов CA - настройка security.selfSigned=false

Для OS service mesh конфигурирование необходимо выполнять вручную, изменяя соответствующие конфиг файлы. Для этого:

Установить следующие значения в конфигурации ServiceMeshControlPlane

istio:

security:

**selfSigned: false**

global:

**mtls:**

**enabled: true**

После обновления деплоя Цитадели убедиться в успешнос запуске и в наличие у нее при запуске следующих логов:

2020-04-13T09:06:18.882849Z info Use certificate from argument as the CA certificate

2020-04-13T09:06:18.965058Z info Creating Kubernetes controller to write issued keys and certs into secret ...

2020-04-13T09:06:19.068232Z info serverCaLog added client certificate authenticator

Для версии Istio 1.5 и выше

istioctl manifest apply --set values.global.mtls.enabled=true,values.security.selfSigned=false

Для версии ниже 1.5

helm template install/kubernetes/helm/istio --name istio --namespace istio-system -x charts/security/templates/deployment.yaml --set global.mtls.enabled=true --set security.selfSigned=false > $HOME/citadel-plugin-cert.yaml

kubectl apply -f $HOME/citadel-plugin-cert.yaml

Если в Istio ранее был сконфигурирован mTLS на основе собственных сертификатов, то необходимо удалить секрет используемый по умолчанию “istio.default”, что бы обновить на приложениях сертификаты принудительно.

oc delete secret istio.default

Проверяем статус Цитадели:

oc get deploy -l istio=citadel -n istio-system

Значения DESIRED/CURRENT/UP-TO-DATE/AVAILABLE должны совпадать.

! Возможно потребуется рестарт кластера, т.к. на локальной инсталяции наблюдались проблемы с деплоем новых конфигов, например Gateway. Ошибка была связана с проблемой проверки сертификатов и решалась рестартом. Выянить, рестарт какого отдельного компонента может решить проблему…

Internal error occurred: failed calling webhook "pilot.validation.istio.io": Post https://istio-galley.istio-system.svc:443/admitpilot?timeout=30s: x509: certificate signed by unknown authority

**3. Установка политики**

Нам необходимо установить политику безопастности правила машрутизации при которых все взаимодействия между сервисами в service-mesh выполняются по защищенному каналу – mTLS. Соответствующие конфигурационные файлы приложены (mesh\_destinationRule\_strict.yaml и mesh\_policy\_strict.yaml)

oc apply -f mesh\_destinationRule\_strict.yaml

oc apply -f mesh\_policy\_strict.yaml

Проверяем текущее значение Policy для service-mesh

oc get meshpolicies -o yaml | grep ' mode'

Для OpenShift Service mesh команда должна быть следующей:

oc get servicemeshpolicies -o yaml | grep ' mode'

Результат выполнения команды должен быть следующим:

"mode: STRICT"

Так же возможно, что конфиг содержит значение аналогичное "mode: STRICT"

spec:

peers:

- mtls: {}

В этом случае предыдущая команда команда «servicemeshpolicies» не выведет ни чего. Нужно выполнить команду следующуб команду и убедиться в наличие значения «mtls: {}»

oc get servicemeshpolicies -o yaml

**Проверки**

1) Проверяем использование новых сертификатов.

Для проверки нам потребуется любое приложение, установленное в кластер.

* Сохраняем имя (**POD\_NAME**) пода в переменную

TEST\_APP\_POD\_NAME=**POD\_NAME**

* Получаем сертификаты из Istio sidecar-proxy этого пода

1) oc exec -it $TEST\_APP\_POD\_NAME -c istio-proxy -- /bin/cat /etc/certs/root-cert.pem > /tmp/pod-root-cert.pem

2) oc exec -it $TEST\_APP\_POD\_NAME -c istio-proxy -- /bin/cat /etc/certs/cert-chain.pem > /tmp/pod-cert-chain.pem

* Проверяем, что корневой сертификат пода и Цитадели(сконфигурированный нами на шаге №2) совпадают. Команда diff должна не должна ни чего вывести.

1) openssl x509 -in custom/certs/root-cert.pem -text -noout > /tmp/root-cert.crt.txt

2) openssl x509 -in /tmp/pod-root-cert.pem -text -noout > /tmp/pod-root-cert.crt.txt

3) diff /tmp/root-cert.crt.txt /tmp/pod-root-cert.crt.txt

* Проверяем, что CA сертификат пода и Цитадели(сконфигурированный нами на шаге №2) совпадают. Команда diff должна не должна ни чего вывести.

1) sed '0,/^-----END CERTIFICATE-----/d' /tmp/pod-cert-chain.pem > /tmp/pod-cert-chain-ca.pem

2) openssl x509 -in custom/certs/ca-cert.pem -text -noout > /tmp/ca-cert.crt.txt

3) openssl x509 -in /tmp/pod-cert-chain-ca.pem -text -noout > /tmp/pod-cert-chain-ca.crt.txt

4) diff /tmp/ca-cert.crt.txt /tmp/pod-cert-chain-ca.crt.txt

* Проверяем, что цепочка сертификатов пода и Цитадели(сконфигурированный нами на шаге №2) совпадают. Команда diff должна не должна ни чего вывести.

1) head -n 21 /tmp/pod-cert-chain.pem > /tmp/pod-cert-chain-workload.pem

2) openssl verify -CAfile <(cat custom/certs/ca-cert.pem custom/certs/root-cert.pem) /tmp/pod-cert-chain-workload.pem

Если в service-mesh развернуто приложение, принимающее или выполняющее запросы по http, то можно убедиться в том, что используется защищенное(https) соединение используя команду tcpdump.

Пример:

- в отдельном терминале определяем имя пода приложения (**POD\_NAME**)

APP\_POD=**POD\_NAME**

oc exec -it $APP\_POD -c istio-proxy /bin/bash

Определяем IP интерфейса eth0

ifconfig

В данном примере IP интерфейса 172.17.0.8

Вывовдим трафик на 8080(либо любой другой указанный в сервисе) в консоль

sudo tcpdump -vvvv -A -i eth0 '((dst port 8080) and (net 172.17.0.8))'

В отдельном терминале выполняем запрос так, что бы в итоге вызов дошел до прослушиваемого сервиса. Запрос обязательно должен пройти через cidecar-proxy прослушиваемого сервиса, лобо посредством обращения через промежуточный сервис в srvice-mesh, либо через Ingres Istio.

В терминале с tcpdump мы должны видеть контент, аналогичный приведенному ниже:

E....T@.@.\................@..X....A[......

.PLo.O......G.........}...,....;9'..:..?....@SLb1.........B|.3.zT;.o.P..M...Q......:."-.\_E.."q.....|...;.G....W.P.....a...D.X0f........4\~<......v`.wq...A.Q#..:...z.P|n.W+...+p....?.

\

.f.}t.t..>hv...2;.5..c$l......).d...%...?a...1..!.P.c.........8..yP.p...(t.^........-....VP..$h....w...y.V.GG....%.]..... +GgF...c..\....Xv........#.q..yR;....u... ..&...9b..h....4..,T...F.+....f....M..Y..[/g..m..7..O....I.1%.C. ....

f%."c.-.@N..J.F~Bm.S.p...mH.\_..2`3..].S2 .^S..|..G..[...\_...43`..V3.c.n..@...=T.

E.....tW.8.p. j..Am]2....m...P.[........s.G.'...T..eN.......o....D..o0?.{|....G1..u.Ac\_..3..!!.v.fY.g

.=.B..\_.j.^..].X.Y;?w......8....Z.....F...w............y1:.P..=..P.8.dYf.5...g.d.f..uc1\*1.[$.K...#....CS.^.p.!V.N...].B...Z..B.......P..qW.-.Rq.ey.e.Iq.:.a..Q.h.?...gu.........~......pb..:RN%7w.&.......w...f......\*..K.0H....C.4'.s.v.De.....(..@..s.#F

12:09:13.268279 IP (tos 0x0, ttl 64, id 33365, offset 0, flags [DF], proto TCP (6), length 52)