作者：李晓辉

联系方式：

1. 微信：Lxh_Chat

2. 邮箱：939958092@qq.com

零信任环境概念

零信任环境就是每次交互都先当它是不受信任的，这样更安全。用户只能访问那些明确被允许的文件或对象，通信还得加密，客户端应用也得验证服务器是不是真的靠谱。而且，零信任环境要求用受信任的证书颁发机构（CA）签名的证书来加密流量，通过引用 CA 证书，应用就能用已签名的证书加密验证另一个应用是不是真的。

service-ca控制器

OpenShift 有个 `service-ca` 控制器，挺有意思的。它能为内部流量生成并签名服务证书，还会创建一个机密，里面装着已签名的证书和密钥。Deployment 可以把这个机密挂载成卷，然后就能用上已签名的证书啦。不过，客户端应用也得信任 `service-ca` 控制器的 CA 哦。

`service-ca` 证书有效期默认是 26 个月，13 个月后会自动轮转。轮转完还有 13 个月的宽限期，原来的 CA 证书在这期间还能用。要是有 pod 是信任老的 CA 证书的，那在宽限期内得找个机会重启一下，重启后服务会自动把新的 CA 捆绑包加进去。

要是你想手动来轮转证书，操作也挺简单。要是想轮转生成的服务证书，就把现有的机密删掉，service-ca 控制器会自动给你生成一个新的。

```
oc delete secret certificate-secret
```

要是想手动轮转服务 CA 证书，那就把 openshift-service-ca 命名空间里的 signing-key 机密删掉。不过要注意，这个操作会让之前的服务 CA 证书马上失效，你得赶紧把所有用到它的 pod 都重启一下，不然 TLS 就没法正常工作了。

```
oc delete secret/signing-key -n openshift-service-ca
```

零信任案例

要是想生成证书和密钥对，只要给服务加上 `service.beta.openshift.io/serving-cert-secret-name=your-secret` 这个注释就行啦。`service-ca` 控制器会瞅瞅同一命名空间里有没有 `your-secret` 这个机密，要是没有，就直接创建一个，然后把服务的已签名证书和密钥对都塞进去。

创建后端服务

比如说，我们来为服务生成一个包含证书对的机密：lxh-secret

第一步，先把服务创建出来

```
oc project default
oc new-app --name hello --image registry.ocp4.example.com:8443/redhattraining/hello-world-n
oc get service

NAME              TYPE           CLUSTER-IP       EXTERNAL-IP                                      PO
hello             ClusterIP      172.30.141.226   <none>                                           80
```

这个8080端口不方便访问，我们直接给它删了，改成443，下面的8888端口，是我们后期打算让pod工作在这个端口

```
oc delete service hello
oc expose deployment/hello --port 443 --target-port 8888
```

```
[student@workstation ~]$ oc get service hello
NAME    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
hello   ClusterIP   172.30.83.38   <none>        443/TCP   105s
```

为服务创建tls机密

第二步，为服务启用证书填充

```
[root@workstation ~]# oc annotate service hello service.beta.openshift.io/serving-cert-secr
service/hello annotate
```

看看服务是否添加了这个注解，看上去多了3个cert相关的注解

```
[root@workstation ~]# oc describe service hello
Name:                   hello
Namespace:              default
Labels:                 app=hello
                        app.kubernetes.io/component=hello
                        app.kubernetes.io/instance=hello
Annotations:            openshift.io/generated-by: OpenShiftNewApp
                        service.alpha.openshift.io/serving-cert-signed-by: openshift-service-ser
                        service.beta.openshift.io/serving-cert-secret-name: lxh-secret
                        service.beta.openshift.io/serving-cert-signed-by: openshift-service-serv
```

我们来研究一下，这个自动生成的机密到底是什么

```
[root@workstation ~]# oc describe secrets lxh-secret
Name:         lxh-secret
Namespace:    default
Labels:       <none>
Annotations:  service.alpha.openshift.io/expiry: 2026-12-20T06:39:38Z
              service.beta.openshift.io/expiry: 2026-12-20T06:39:38Z
              service.beta.openshift.io/originating-service-name: hello
              service.beta.openshift.io/originating-service-uid: fc4aafa0-fa9f-4b7b-9d44-e5

Type:  kubernetes.io/tls

Data
====
tls.crt:  2575 bytes
tls.key:  1675 bytes
```

证书是啥内容看看去

```
[root@workstation ~]# oc get secrets lxh-secret -o yaml
apiVersion: v1
data:
  tls.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR3VENDQXFtZ0F3SUJBZ0lJQVdmMURucWtFSmN3
  tls.key: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBb011NGhDU2JvYmE3
kind: Secret
metadata:
  annotations:
    service.alpha.openshift.io/expiry: "2026-12-20T06:39:38Z"
    service.beta.openshift.io/expiry: "2026-12-20T06:39:38Z"
    service.beta.openshift.io/originating-service-name: hello
    service.beta.openshift.io/originating-service-uid: fc4aafa0-fa9f-4b7b-9d44-e52507b893a4
  creationTimestamp: "2024-12-20T06:39:38Z"
  name: lxh-secret
  namespace: default
  ownerReferences:
  - apiVersion: v1
    kind: Service
    name: hello
    uid: fc4aafa0-fa9f-4b7b-9d44-e52507b893a4
  resourceVersion: "170877"
  uid: 51726655-b1cb-407b-8c76-0f2551635b1e
type: kubernetes.io/tls
```

可以看到这个自动生成的机密中，的确包括了两个证书，我们只需要把证书挂载到我们的deployment中就可以了，我们先更新一下配置文件来包含tls部分

为后端服务准备tls配置文件

我们要读取的证书位于：/etc/pki/nginx/，让pod工作在8888端口

```
cat > nginx.conf <<-'EOF'
server {
    listen       8888 ssl http2 default_server;
    listen       [::]:8888 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_ciphers PROFILE=SYSTEM;
    ssl_prefer_server_ciphers on;

    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
        location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
        location = /50x.html {
    }
}
EOF
```

把这个配置文件存为configmap，一会儿让deployment来挂载

```
[student@workstation ~]$ oc create configmap lxh-nginx-tls --from-file nginx.conf
configmap/lxh-nginx-tls created
```

用补丁文件的方法修改，比手工改更精准

```
cat > patch.yml <<-'EOF'
spec:
  template:
    spec:
      containers:
        - name: hello
          ports:
            - containerPort: 8888
          volumeMounts:
            - name: tls-config
              mountPath: /etc/nginx/conf.d/
            - name: server-secret
              mountPath: /etc/pki/nginx/
      volumes:
        - name: tls-config
          configMap:
            defaultMode: 420
            name: lxh-nginx-tls
        - name: server-secret
          secret:
            defaultMode: 420
            secretName: lxh-secret
            items:
              - key: tls.crt
                path: server.crt
              - key: tls.key
                path: private/server.key
EOF
```

来，打一下补丁，发现我们的pod age很新，是刚创建的

```
[student@workstation ~]$ oc patch deployment hello --patch-file patch.yml
[student@workstation ~]$ oc get pod
NAME                          READY   STATUS             RESTARTS       AGE
hello-74fbfc9fd-ppzpq         1/1     Running            0              28s
```

测试陌生pod访问

我们随便找个调试pod，看看能不能访问service

```
[student@workstation ~]$ oc get service hello
NAME    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
hello   ClusterIP   172.30.83.38   <none>        443/TCP   105s
```

发现，随便找个pod是无法建立安全连接的，但是用-k跳过证书验证可以访问

以下hello.default.svc的格式是k8s自带的

1. hello是服务名称
2. default是命名空间
3. svc表示这是一个服务

```
[student@workstation ~]$ oc debug -t

sh-4.4# curl https://hello.default.svc

curl: (60) SSL certificate problem: self signed certificate in certificate chain
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.

sh-4.4# curl https://hello.default.svc -k
<html>
  <body>
    <h1>Hello, world from nginx!</h1>
  </body>
</html>
```

为客户端准备CA证书

生成包含服务 CA 捆绑包的configmap，并使用它来创建 `client`

```
[student@workstation ~]$ oc create configmap ca-file
configmap/ca-file created
```

给这个configmap填充ca证书

```
[student@workstation ~]$ oc annotate configmap ca-file service.beta.openshift.io/inject-cab
configmap/ca-file annotate
[student@workstation ~]$ oc describe configmaps ca-file
Name:         ca-file
Namespace:    default
Labels:       <none>
Annotations:  service.beta.openshift.io/inject-cabundle: true

Data
====
service-ca.crt:
----
-----BEGIN CERTIFICATE-----
MIIDUTCCAjmgAwIBAgIIFAxCe1I9d3swDQYJKoZIhvcNAQELBQAwNjE0MDIGA1UE
Awwrb3BlbnNoaWZ0LXNlcnZpY2Utc2VydmluZy1zaWduZXJAMTcwNjAxMTc0NDAe
Fw0yNDAxMjMxMjA5MDNaFw0yNjAzMjMxMjA5MDRaMDYxNDAyBgNVBAMMK29wZW5z
aGlmdC1zZXJ2aWNlLXNlcnZpbmctc2lnbmVyQDE3MDYwMTE3NDQwggEiMA0GCSqG
SIb3DQEBAQUAA4IBDwAwggEKAoIBAQDBmQBMDkBshSKSBnnPSDOJFx5lN05uj1ij
QkdMkvZ77UF6+grNK7J2XjMUL7XGV1AB2dylr+/Ze8bv+zgaKVPuz33v5Qkq1Xq3
sGTCcvEOKFFQpNi2xvvz+SRxE3ZepSn466d8Yl8KAMOwUs41SV1hRWhMjDnQpJFY
1o6zBSF3NUHrjwpgdaoqxvpAZq0F12ZdjmP6kY64CvYujUxjpZ+WTwkqQHi/RXfL
F3JkbhX/dmMsMG4lRegMwzcUvrNHV89pqg82urLAXKpEdeaqiDq1rz5ImomTJUyY
nDFDQWApBS+ds++M364pKGktIIJT4S9bp1+HJWBMlVR3yRcglgD1AgMBAAGjYzBh
MA4GA1UdDwEB/wQEAwICpDAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBQbVoak
nlEWPiZHAj6Dv7RltXKfyDAfBgNVHSMEGDAWgBQbVoaknlEWPiZHAj6Dv7RltXKf
yDANBgkqhkiG9w0BAQsFAAOCAQEAS3XqN/+utRKusxadlawdR61lDh4CB8mMrhc6
AVTXqdAaE2j+T4xAXMQWkCAs82UDKuCUK2O+PTf9HrePiKLp3YWi+VoqFUoPiI++
KFl24z+kcOuTatJJdutZ3UN8bk4T24GINBlQNyN2P3HDGQ1FL2NLNc59xC5qxFGC
I1j4RwsxmDz2SIlPeselEMS/unqpWTAW4M9ZkMmvg7BeHsUnNtHJPtQwqYkaWlXn
df5REDU7IkMKuNdlxD5PehUjujGB29PaBubfxBxFlhFyKLWJ72nECEZdJAwku1sf
L+TYgUQf39c9HNo9tBmDg/XdOXGM0BUjDQ/rTo0mcbSmp6b/dg==
-----END CERTIFICATE-----


BinaryData
====

Events:  <none>
```

验证零信任效果

我们来启动一个带有此ca的客户端访问一下看看

```
cat > pod-with-ca.yml <<-'EOF'
apiVersion: v1
kind: Pod
metadata:
  name: client
spec:
  containers:
    - name: client
      image: registry.ocp4.example.com:8443/redhattraining/hello-world-nginx
      volumeMounts:
        - mountPath: /etc/pki/ca-trust/extracted/pem
          name: trusted-ca
  volumes:
    - configMap:
        defaultMode: 420
        name: ca-file
        items:
          - key: service-ca.crt
            path: tls-ca-bundle.pem
      name: trusted-ca
EOF
```

```
oc create -f pod-with-ca.yml
```

可以发现，直接访问成功，不需要-k跳过证书验证了

```
sh-4.4$ curl https://hello.default.svc
<html>
  <body>
    <h1>Hello, world from nginx!</h1>
  </body>
</html>
```

研究一下证书交互过程

```
sh-4.4$ curl https://hello.default.svc -vv -I
* Rebuilt URL to: https://hello.default.svc/
*   Trying 172.30.246.118...
* TCP_NODELAY set
* Connected to hello.default.svc (172.30.246.118) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
  CApath: none
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, [no content] (0):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, [no content] (0):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, [no content] (0):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, [no content] (0):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, [no content] (0):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: CN=hello.default.svc
*  start date: Dec 20 08:36:30 2024 GMT
*  expire date: Dec 20 08:36:31 2026 GMT
*  subjectAltName: host "hello.default.svc" matched cert's "hello.default.svc"
*  issuer: CN=openshift-service-serving-signer@1706011744
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* TLSv1.3 (OUT), TLS app data, [no content] (0):
* TLSv1.3 (OUT), TLS app data, [no content] (0):
* TLSv1.3 (OUT), TLS app data, [no content] (0):
* Using Stream ID: 1 (easy handle 0x561a9eab7a00)
* TLSv1.3 (OUT), TLS app data, [no content] (0):
> HEAD / HTTP/2
> Host: hello.default.svc
> User-Agent: curl/7.61.1
> Accept: */*
>
* TLSv1.3 (IN), TLS handshake, [no content] (0):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
```

```
* TLSv1.3 (IN), TLS handshake, [no content] (0):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS app data, [no content] (0):
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
* TLSv1.3 (OUT), TLS app data, [no content] (0):
* TLSv1.3 (IN), TLS app data, [no content] (0):
* TLSv1.3 (IN), TLS app data, [no content] (0):
< HTTP/2 200
HTTP/2 200
< server: nginx/1.14.1
server: nginx/1.14.1
< date: Fri, 20 Dec 2024 08:52:26 GMT
date: Fri, 20 Dec 2024 08:52:26 GMT
< content-type: text/html
content-type: text/html
< content-length: 72
content-length: 72
< last-modified: Wed, 26 Jun 2019 22:19:37 GMT
last-modified: Wed, 26 Jun 2019 22:19:37 GMT
< etag: "5d13ef79-48"
etag: "5d13ef79-48"
< accept-ranges: bytes
accept-ranges: bytes

<
* Connection #0 to host hello.default.svc left intact
```

也可以用下面的方法验证过程

```
openssl s_client -connect hello.default.svc:443
```

本文档在线版本：https://www.linuxcenter.cn