Achieving Mutual TLS

Ted Hahn, TCB Technologies, Inc.

Mark Hahn, Ciber Global

tcbtech.com/kubetls
Achieving Mutual TLS

Secure Pod-to-Pod communication without the hassle

Every Kubernetes pod should include a SSL Certificate, verifying its identity. This should be signed by the kubernetes master, and be specific to each pod.
Certificate Lifecycle

- Create a private key - public key pair
- Put the public key in a Certificate Signing Request (CSR)
- Certificate Authority (CA) signs the CSR to create a Certificate
  - I.e. The CA computes a hash of the CRS's public key using the CA's private key
- The CA returns the Certificate to the requester

A client without prior knowledge of the server but with prior trust in the Certificate Authority, can communicate with the Server.
### Manual Creation of Certificates

**cfssl**

```bash
# Create a Certificate Authority
cd ca
cfssl genkey -initca ca-csr.json | cfssljson -bare

# Create and then sign the server certificate
cfssl genkey server-csr.json | cfssljson -bare
cfssl sign -ca ../ca/cert.pem -ca-key ../ca/cert-key.pem cert.csr | cfssljson -bare
openssl verify -CAfile ../ca/cert.pem cert.pem

# Create and then sign the client certificate
cd ../client/
cfssl genkey client-csr.json | cfssljson -bare
cfssl sign -ca ../ca/cert.pem -ca-key ../ca/cert-key.pem cert.csr | cfssljson -bare
```
Manual Creation of Certificates

openssl

# create openssl.cnf, tweak details
rm index.txt
touch index.txt
echo 1000 > serial

openssl ecparam -name prime256v1 -genkey > private/ca.key.pem
openssl req -config openssl.cnf -key private/ca.key.pem -new -x509 -extensions v3_ca -out certs/ca.cert.pem -subj "/CN=CA"
openssl x509 -in certs/ca.cert.pem -text

openssl ecparam -name prime256v1 -genkey > dev-key.pem
openssl req -new -key dev-key.pem -out dev.csr -config req.conf
openssl ca -config openssl.cnf -extensions server_cert -in dev.csr -batch -out dev-cert.pem
How are Certificates used

Developers need to include code in their programs/microservices to do the configuration and initialization to serve requests through TLS.
Golang Applications without TLS

```go
var (  
    listenAddr = flag.String("listenAddr", ":8080", "Address to listen on")
)

func main() {
    flag.Parse()
    goApi := goApi.NewGoApi()
    http.HandleFunc("/go/api", goApi.endPoint)
    http.HandleFunc("/webhook", sslmutator.AcceptWebhook)
    log.Printf("Listening on %s", *listenAddr)
    log.Fatal(http.ListenAndServe(*listenAddr, nil))
}
```
Golang Applications using TLS

```go
var (
    listenAddr = flag.String("listenAddr", ":8443", "Address to listen on")
    certFile   = flag.String("certFile", "/var/tls/tls.crt", "TLS Certificate")
    keyFile    = flag.String("keyFile", "/var/tls/tls.key", "TLS Key")
)

func main() {
    log.Printf("Listening on %s", *listenAddr)
    log.Fatal(http.ListenAndServe(*listenAddr, nil))
}
```

```go
func main() {
    http.HandleFunc("/webhook", sslmutator.AcceptWebhook)
    log.Println("Listening on %s", *listenAddr)
}
```
Java Applications without TLS

```java
public static void main(String[] args) throws IOException {
    SpringApplication.run(ApiDemonstration.class, args);
}

@RestController
@RequestMapping("/api")
public class ApiEndpointController {
    @GetMapping("/testinfo")
    public TestInfo getTestInfo() {
        return new TestInfo();
    }
}
```
public static void main(String[] args) throws IOException {
    System.setProperty("server.ssl.enabled", "true");
    System.setProperty("server.ssl.key-store", "/var/tls/tls.p12");
    System.setProperty("server.ssl.key-store-password", "abc123");
    SpringApplication.run(ApiDemonstration.class, args);
}

@RestController
@RequestMapping("/api")
public class ApiEndpointController {
    @GetMapping("/testinfo")
    public TestInfo getTestInfo() {
        return new TestInfo();
    }
}
Mutual TLS in your services

- So far configuring the certificates is easy, but getting the certificate is the hard part
- Generating and storing the certificates correctly is fiddly
- Error reporting from certificate tools, and programming languages is poor
- How can we make this easier?
TLS keys with KubeTLS

Kubernetes to the rescue!

- Kubernetes already has a built-in Certificate Authority
- One could make the case that a Kubernetes cluster is defined by its master certificate authority
- Kubernetes should create TLS certificates for all pods/services/deployments
- Similar to Borg/ALTS
The plan:

- Create a TLS cert that is compatible with the way you use Kubernetes services already
- Generate a certificate and key for every service (endpoint)
- Provide that certificate and key to every pod that matches that service
- All certificates are automatically generated by a central (Kubernetes) CA, and trusted by everything else in the cluster
The details:

Mutating Controller

- Kubernetes provides a [webhook interface](#) which for verifying and modifying objects as they are created
- KubeTLS is our service which implements a webhook watching for pod creation
- Pod create requests are modified to included a TLS secret with that POD's certificate and key
KubeTLS in Action

Demo Time!
Mutating Webhook Request Body

```json
{
  "kind": "AdmissionReview",
  "apiVersion": "admission.k8s.io/v1beta1",
  "request": {
    "uid": "43a278cc-6ee3-4f6c-a1bd-7c2ca21d7df4",
    "kind": {
      "group": "",
      "version": "v1",
      "kind": "Pod"
    },
    "resource": {
      "group": "",
      "version": "v1",
      "resource": "pods"
    },
    "requestKind": {
      "group": "",
      "version": "v1",
      "kind": "Pod"
    },
    "requestResource": {
      "group": "",
      "version": "v1",
      "resource": "pods"
    },
    "namespace": "default",
    "operation": "CREATE",
    "userInfo": {
      "username": "system:serviceaccount:kube-system:replicaset-controller",
      "uid": "f80d5e0e-e75f-4c42-adb5-f49231d03278",
      "groups": [
        "system:serviceaccounts",
        "system:serviceaccounts:kube-system",
        "system:authenticated"
      ]
    },
    "object": {
      "kind": "Pod",
      "apiVersion": "v1",
      ...
    }
  }
}
```
Mutating Webook Response Body

```
},
"response": {
  "uid": "43a278cc-6ee3-4f6c-a1bd-7c2ca21d7df4",
  "allowed": true,
  "status": {
    "metadata": {}, "status": "Success"
  },
  "patch": "W3sib3AiOiJh...HMtYmMxYzhhMWY1LCJzZWNoZXQ1Onsic2VjcmV0TmFtZSI6InRscy1iYzFjOGExZI9fX1d",
  "patchType": "JSONPatch"
}
```

```json
- - - - - - - - - - - - - -
[
  { "op": "add",
    "path": "/spec/containers/0/volumeMounts/-",
    "value": {
      "mountPath": "/var/run/secrets/gauntletwizard.net/tls",
      "name": "tls-94d67ccf" } },
  { "op": "add",
    "path": "/spec/volumes/-",
    "value": {
      "name": "tls-94d67ccf",
      "secret": {
        "secretName": "tls-94d67ccf"
      } }
] ```
func (s TLSController) AcceptWebhook(w http.ResponseWriter, r *http.Request) {
    body, err := ioutil.ReadAll(r.Body)
    admissionRequest, err := parseAdmissionRequest(body)
    pod := &corev1.Pod{}
    pod.Namespace = admissionRequest.Request.Namespace
    services, err := s.ss.MatchingServices(r.Context(), pod)
    secret, err := s.tlss.SecretForServices(r.Context(), *pod, services)
    review := Mutate(*admissionRequest, secret.Name)
    response, err := json.Marshal(review)
    w.WriteHeader(200)
    w.Write(response)
}
Create And Upload Csr

```go
csrObject := certsv1.CertificateSigningRequest{}
csrObject.Name = info.Name()
csrObject.Annotations = info.Annotations()
csrObject.Labels = t.labels
```

```go
csrObject.Spec.SignerName = &SignerName
```

```go
csrObject.Spec.Request = csr
```

```go
// Allowed usages of our cert.
```

```go
createdCsrObject, err := client.Create(ctx, &csrObject, metav1.CreateOptions{})
```

```go
// This is the stamp we assign to the CSR object to let the cluster master sign it.
approval := certsv1.CertificateSigningRequest{
    Status: certsv1.CertificateSigningRequestStatus{
        Conditions: []certsv1.CertificateSigningRequestCondition{
            Type: certsv1.CertificateApproved,
            LastUpdateTime: metav1.Now(),
        },
    },
}
```

```go
approval.ObjectMeta = createdCsrObject.ObjectMeta
```

```go
approved, err := client.UpdateApproval(ctx, &approval, metav1.UpdateOptions{})
```
secretToPass := &corev1.Secret{
    Name = info.Name(),
    Annotations = info.Annotations(),
    Labels = t.labels,
    Type = corev1.SecretTypeTLS,
}

keybytes, key, csr := csrForServices(info)

if err != nil {
    fmt.Println("Failed to convert to Pkcs12 store, ", err)
    return nil, err
}

secretToPass.Data = map[string][]byte{
    "tls.key": key,
    "tls.csr": csr,
    "tls.crt": cert,
    "pkcs12.certStore": pkcs12store,
    "pkcs12.password": pkcs12password,
}

pkcs12store, pkcs12passord, err := t.convertToPkcs12(cert, &keybytes)
Deploy Go App with MTLS

- Simple golang grpc
- Job communicating via mutual tls
- Pods have the correct keys applied automatically
Using KubeTLS

```go
cost {
    KubeTLSKeyLocation = KubeTLSectLocation + "tls.key"
    KubeTLSCertLocation = KubeTLSectLocation + "tls.crt"
    KubeTLSCLocation = "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt"
}

// NewKubeTLS constructs a TLS Configuration for use with KubeTLS, suitable for use in both client and server
func NewKubeTLS() (*tls.Config, error) {
    certificate, err := tls.LoadX509KeyPair(KubeTLSectLocation, KubeTLSKeyLocation)
    ...
    serverpool, err := x509.SystemCertPool()
    serverpool.AppendCertsFromPEM(rootCertData)
    return &tls.Config{
        Certificates: [][]tls.Certificate{certificate},
        // Server options
        ClientAuth:   tls.RequireAndVerifyClientCert,
        ClientCAs:    clientpool,
        // Client options
        RootCAs: serverpool,
    }
```
Using KubeTLS

- libraries in golang
- libraries in java
- more to come . . .
Compare to Service Mesh

- This is service Mesh
- Istio adds two contact edges - Client <-> Mesh, Mesh <-> Server
- Debugging is hard - Harder even than TLS
- Abstracts too much from Developers
Future Directions

- More languages supported
- Ideally this would be in the Kubelet
- Changes in 1.19 alter they way that this works
Repository

https://gitlab.com/gauntletwizard_net/kubetls

Presentation

tcbtech.com/kubetls
Questions?
TLS Secret Creation
Use Kubernetes as a signing server manually

```
# Certificate authority is in the Kubernetes API

cfssl genkey server-csr.json | cfssljson -bare

echo {} | jq --rawfile csr cert.csr '{apiVersion: "certificates.k8s.io/v1beta1", kind: "CertificateSigningRequest", metadata: {name: "kubetls-bootstrap"}, spec: {request: $csr|@base64, usages: ["server auth", "client auth", "digital signature", "key encipherment"]}}' | kubectl create -f

kubectl certificate approve kubetls-bootstrap

kubectl get csr kubetls-bootstrap -o json | jq -.status.certificate | @base64d > cert.crt
```