Securing your gRPC Application

Authentication, Authorization, and RBAC in gRPC

November 21 2019

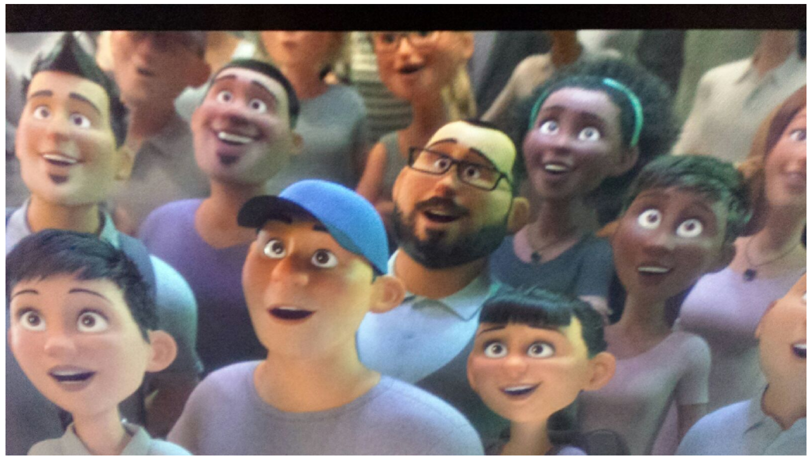
Luis Pabón
MTS, Portworx

About me

Luis Pabón

- CNCF Storage Technical Lead
- Kubernetes SIG-Storage Community Member
- Container Storage Interface (CSI) Community Member

Previously at CoreOS and Red Hat Storage



History

Requirements:

- We wanted to create an SDK to make it easy for developers to integrate Portworx technology with their control plane
- We wanted to make sure that only certain users had the ability to use certain resources

We createad the OpenStorage SDK(https://libopenstorage.github.io), a gRPC based service which supports authentication and authorization with RBAC.

This talk is based on our experience creating this service.

Security Models

Authentication:

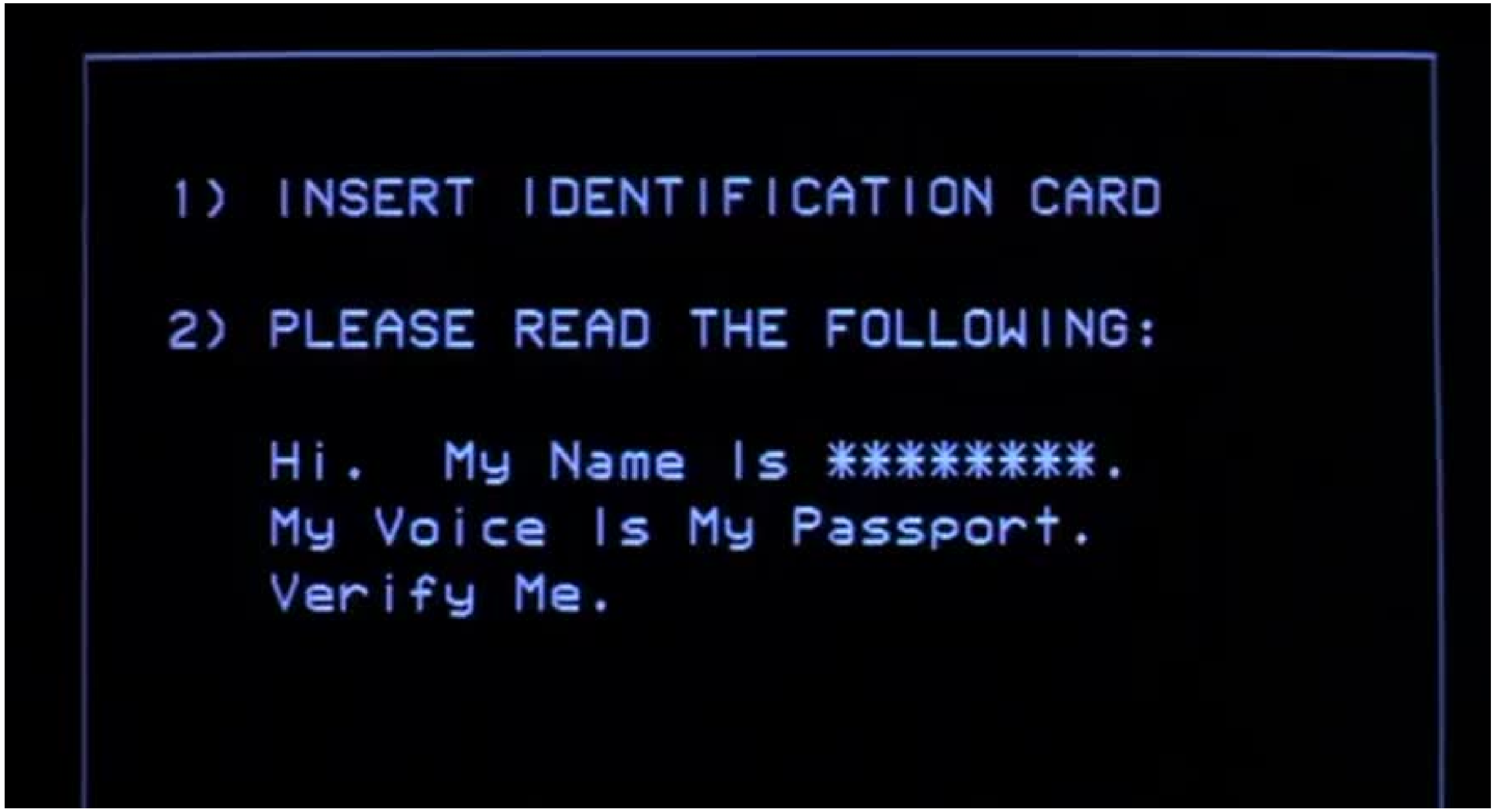
- Who are you?
- How can I trust that you are who you say you are?
- What other information is there about you?

Authorization:

- Are you allowed to do what you are asking?
- Are you allowed to access that information?

Authentication

Hello, my name



1) INSERT IDENTIFICATION CARD

2) PLEASE READ THE FOLLOWING:

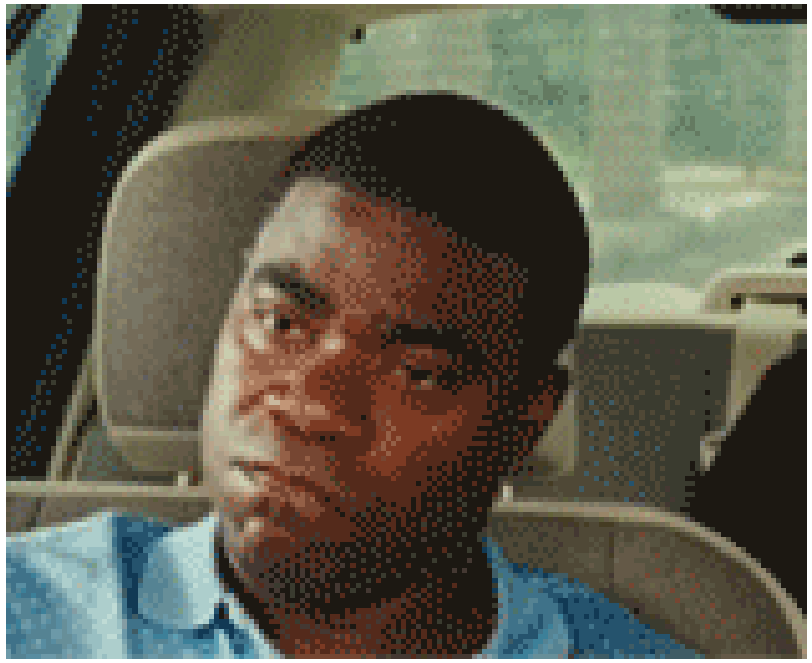
Hi. My Name Is ********.
My Voice Is My Passport.
Verify Me.

6

Security Architecture

- Discuss how to store passwords security
- Manage passwords
- ...

DO NOT DO PASSWORD MANAGEMENT!



Important: Tokens are created by other entities and the gRPC applications only need to **verify** the token

Authentication in gRPC

This talk will be basing authentication on the following models:

- Using JSON Web Token as stated in the IETF draft (https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-25) to identify a user
- gRPC applications should only verify the token from a trusted issuer

JWT

JWT has the following components:

```
[ header . claims . signature ]
```

Example:

```
eyJhb[...omitted for brevity...]HgQ
```

Header: Token and signature types in clean text
Claims: JSON formatted metadata about the user in clear text
Signature: Signature created using crypto hash

Token Authority

A token authority is the issuer of tokens.

Many ways to generate tokens, but there are mainly two:

- An application generates a token
- OpenID Connect compliant server to generate tokens

Application to generate a token

Golang pseudocode

```
mapclaims := jwt.MapClaims{
  "sub":   claimsSubject,
  "iss":   tokenIssuer
  "email": claimsEmail,
  "name":  claimsName,
  "role":  claimsRole,
  "iat":   time.Now().Unix(),
  "exp":   time.Now().Add(expDuration).Unix(),
}
token := jwt.NewWithClaims(signature.Type, mapclaims)
signedtoken, err := token.SignedString(signature.Key)
```

See github.com/libopenstorage/openstorage-sdk-auth (https://github.com/libopenstorage/openstorage-sdk-auth/blob/892edc04561b26f6531fdda4383b2e0da55cc789/pkg/auth/auth.go#L57-L70)

12

OpenID Connect

Using an application to create tokens may satisfy many deployments, but some may require management of thousands of users.

In this scenario, managing users is easier through a OpenID Connect, ODIC, compliant system.

OIDCs:

- Keycloak (open source) (https://www.keycloak.org/)
- Dex (open source) (https://github.com/dexidp/dex)
- OpenUnison (open source) (https://github.com/tremolosecurity/openunison)
- Okta (https://www.okta.com/)
- Auth0.com (https://auth0.com/)
- Google, Aws, etc.

13

Client Token

Clients insert the the token in the gRPC metadata

Example (Golang):

```
import "google.golang.org/grpc/metadata"

md := metadata.New(map[string]string{
    "authorization": "bearer" + token,
  })
ctx = metadata.NewOutgoingContext(context.Background(), md)
_, err := YourGrpcApi(ctx, ...)
```

See: Example (https://github.com/libopenstorage/libopenstorage.github.io/blob/7727f6a7755a4a8c376adf258f760b0801c2eeb9/examples/golang/main.go#L33-L62)

Example (Python):

```
md = []
md.append(("authorization", "bearer "+token))

# Now add metadata to the call
stub = api_pb2_grpc.YourAPIStub(channel)
response = stub.YourApi(api_pb2.YourApiRequest(), metadata=md)
```

gRPC Server Architecture

Use gRPC interceptors to get authentication and authorization support

```
+----------------+
|  Interceptor  |  <----- Client
+----------------+
        |
+----------------+
|   gRPC Server |
|    Handlers   |
+----------------+
```

A simple interceptor

```
func SimpleIntercepter(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler,
) (interface{}, error) {

    // ctx has metadata about the call

    // You can add information in the ctx for other interceptors to use
    ctx = context.WithValue(ctx, "somekey", somedata)

    // info has the API name
    logger.Printf("In SimpleInterceptor: Method=%s", info.FullMethod)

    // Call the next handler
    return handler(ctx, req)
}
```

Interceptors are initialized in the gRPC server configuration.

Authentication registration

Setup the interceptors in order in *ServerOption*

```
import (
  grpc_middleware "github.com/grpc-ecosystem/go-grpc-middleware"
  grpc_auth "github.com/grpc-ecosystem/go-grpc-middleware/auth"
)

opts := make([]grpc.ServerOption, 0)
opts = append(opts, grpc.UnaryInterceptor(
    grpc_middleware.ChainUnaryServer(
        simpleinterceptor,
        grpc_auth.UnaryServerInterceptor(auth),
    )))
grpcServer := grpc.NewServer(opts...)
...
```

See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/api/server/sdk/server.go#L422-L451)

Authentication interceptor

```
import (
  grpc_auth "github.com/grpc-ecosystem/go-grpc-middleware/auth"
  "google.golang.org/grpc/codes"
  "google.golang.org/grpc/status"
)
...
func  auth(ctx context.Context) (context.Context, error) {
    // grpc_auth.AuthFromMD will extract the token from the key
    // "authorization" and return the token after removing the "bearer " prefix
    token, err := grpc_auth.AuthFromMD(ctx, "bearer")
    if err != nil {
        return nil, err
    }
    if err := verify(token); err != nil {
        return nil, status.Errorf(codes.PermissionDenied, err.Error())
    }
    return ctx, nil
}
```

See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/api/server/sdk/server_interceptors.go#L55-L88)

Golang verification libraries

Signed by an application:

- github.com/dgrijalva/jwt-go (github.com/dgrijalva/jwt-go)
- See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/pkg/auth/selfsigned.go#L91)

OIDC:

- github.com/coreos/go-oidc (https://github.com/coreos/go-oidc)
- See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/pkg/auth/oidc.go#L77-L96)

Authorization

RBAC

Role Based Access Control (RBAC) is a model used to authorize user access.

- Kubernetes uses RBAC (https://kubernetes.io/docs/reference/access-authn-authz/rbac/#api-overview) to control access to its API
- In OpenStorge SDK (https://libopenstorage.github.io/w/release-6.1.generated-api.html#serviceopenstorageapiopenstoragerole) we use RBAC to control access to the gRPC API.

Roles are *keys* to *rules*:

RBAC in gRPC

Kubernetes RBAC rules are based on HTTP-like verbs like *get*, *list*, *patch*, etc.
In gRPC we need to do something different.

A gRPC RPC call looks like the following:

```
service RouteGuide {
    rpc GetFeature(Point) returns (Feature) {}
    rpc ListFeatures(Rectangle) returns (stream Feature) {}
}
```

See route_guide.proto (https://github.com/grpc/grpc-go/blob/f7de2c8d62aff2193c58a25252ea5cd183fd26b7/examples/route_guide/routeguide/route_guide.proto#L24)

RBAC in gRPC can be broken down to a set of rules on *Services* and *Apis*.

RBAC in gRPC

Logically, we need the gRPC to support the following:

```
"myrole": [
    "services" : [
        "routeguide"
    ],
    "apis" : [
        "getfeature"
    ]
]
```

In this example, the role *myrole* does not have access to the *ListFeatures()* API.

RBAC Values

- How do we know what string values to pick?
- How do we implement this?

Interceptors have `info.FullMethod` which have the following:

```
/<gRpc Name>Service/Api
```

In OpenStorage SDK we have the following `info.FullMethod`:

```
/openstorage.api.OpenStorage<service>/<Api>
```

See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/pkg/role/sdkserviceapi.go#L338-L339)

Example Rules in Golang

The following is an example from OpenStorage SDK:

```
// "myrole" allow access to all APIs in the Volume and Cluster services
defaultRoles = map[string][]*api.SdkRule{
  "myrole": {
    &api.SdkRule{
        Services: []string{
            "volume",
            "cluster",
        },
        Apis: []string{"*"},
  },
}
```

See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/pkg/role/sdkserviceapi.go#L39-L65)

Authorization Interceptor

```
func authorizationServerInterceptor(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler,
) (interface{}, error) {

    // Get user information place here from the authentication interceptor
    userinfo, ok := auth.NewUserInfoFromContext(ctx)
    claims := &userinfo.Claims

    // Authorize passing in the roles from the JWT claims
    if err := Verify(ctx, claims.Roles, info.FullMethod); err != nil {
        return nil, status.Errorf(codes.PermissionDenied,
            "Access to %s denied: %v", info.FullMethod, err)
    }

    // Execute the command
    return handler(ctx, req)
}
```

See Example (plus Audit log!)

(https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/api/server/sdk/server_interceptors.go#L117)

What about normal REST access to the service?

grpc-gateway (https://github.com/grpc-ecosystem/grpc-gateway)

- Automate generation of a REST to gRPC gateway
- Add an annotation to your proto RPC APIs
- Generate the gateway Golang file
- The gateway will automatically forward the *Authorization* bearer token header
- Just register the handlers and start it

Sample annotation on a proto file

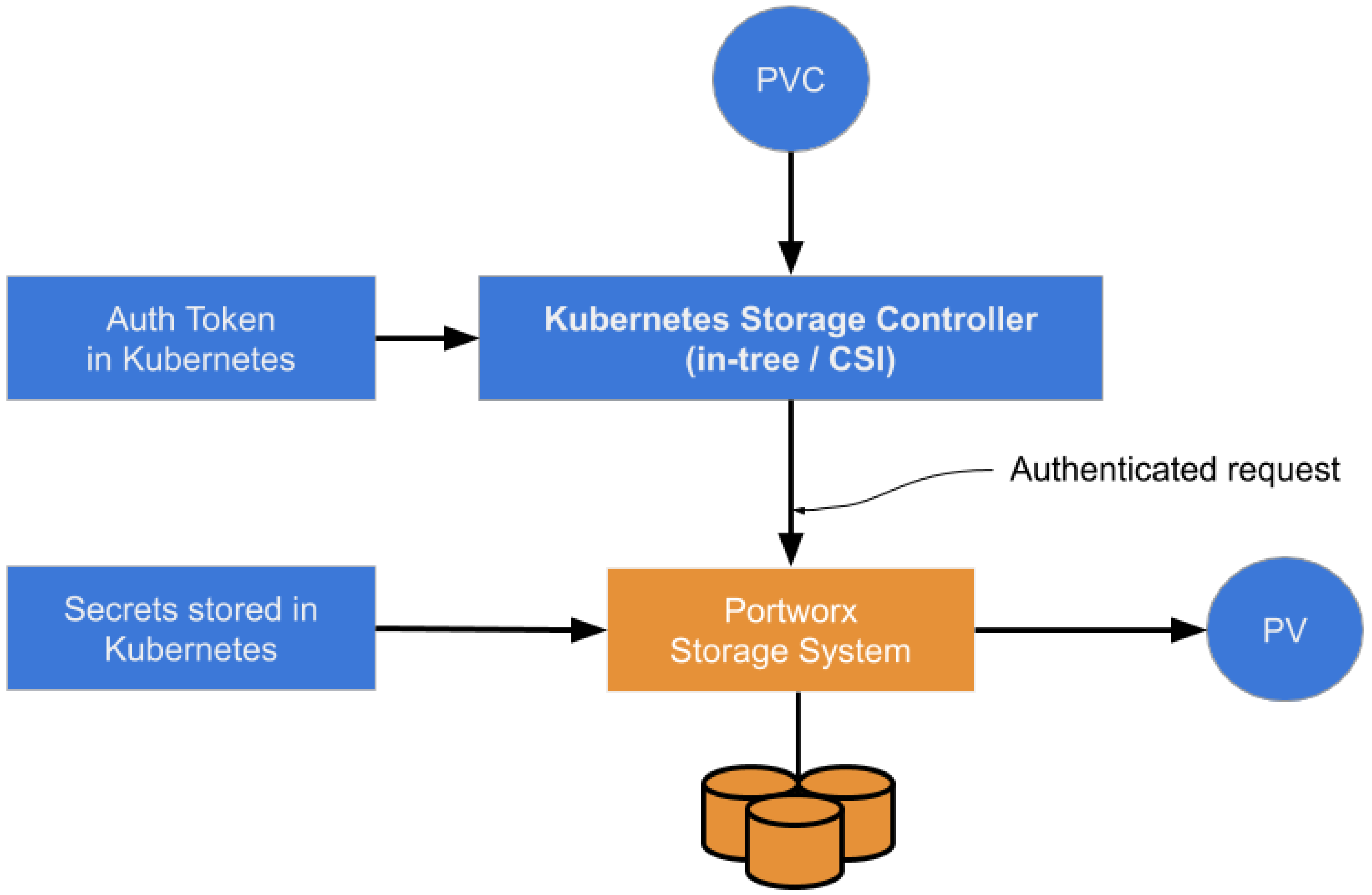
```
service OpenStorageRole {

// Create a role for users in the system
rpc Create(SdkRoleCreateRequest)
    returns (SdkRoleCreateResponse){
        option(google.api.http) = {
            post: "/v1/roles"
            body: "*"
        };
    }
}
```

See Example (https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/api/api.proto#L1084-L1093)

Live Demo!

Demo

Authenticating storage access in Kubernetes


PVC
Auth Token in Kubernetes
Kubernetes Storage Controller (in-tree / CSI)
Authenticated request
Secrets stored in Kubernetes
Portworx Storage System
PV

Some last notes on gRPC

Generate documentation: protoc-gen-doc (https://github.com/pseudomuto/protoc-gen-doc)

- Example: github.com/libopenstorage/libopenstorage.github.io (https://github.com/libopenstorage/libopenstorage.github.io)

Versioning gRPC: Versioning in your proto file

(https://github.com/libopenstorage/openstorage/blob/3d7c200148a18d9586811f0250b1b90f7466e69b/api/api.proto#L3347-L3379)

Thank you

November 21 2019

Luis Pabón
MTS, Portworx
luis@portworx.com (mailto:luis@portworx.com)
https://github.com/lpabon/go-slides (https://github.com/lpabon/go-slides)
Twitter: `@_lpabon_` (#ZgotmplZ)