Kubernetes Auth

KubeCon Europe 2019
Sig-Auth Deep Dive

Mo Khan & Matt Rogers, Red Hat

Presenters

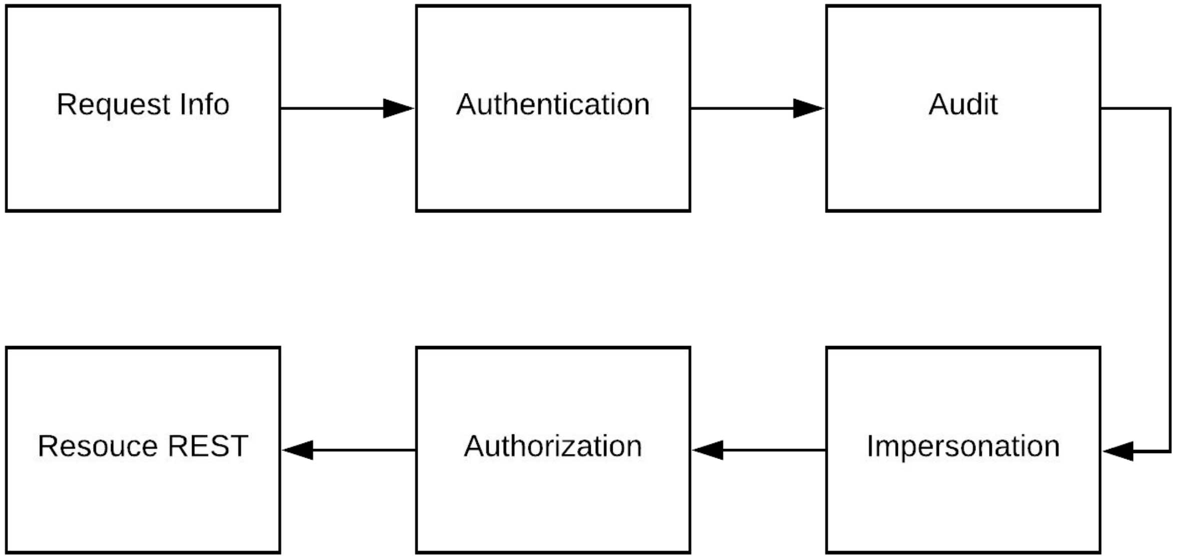
Mo Khan
Software Engineer, Red Hat
https://monis.app
github: @enj

Matt Rogers
Senior Software Engineer, Red Hat
https://mrogers950.gitlab.io
github: @mrogers950

Outline

- Request flow
  - Request handler registration
  - Request context
  - Request metadata
  - Request handler chain
- Authentication
  - Authenticator types
  - Authenticator union
  - Examples
- Authorization
  - Authorizer types
  - Authorizer union
  - Examples

Kubernetes request flow (simplified)



Anatomy of Go HTTP server

```
    type Handler interface {
        ServeHTTP(ResponseWriter, *Request)
    }

----------------------------------------------

    helloHandler := func(w http.ResponseWriter, req *http.Request) {
        io.WriteString(w, "Hello, world!\n")
    }
    http.HandleFunc("/hello", helloHandler)
    log.Fatal(http.ListenAndServe(":8080", nil))
```

Kubernetes request flow

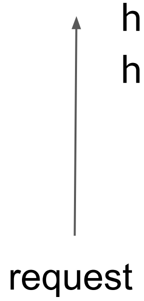
- Layered approach - a series of wrapped http.Handlers

```
func(http.Handler, ...) http.Handler
```

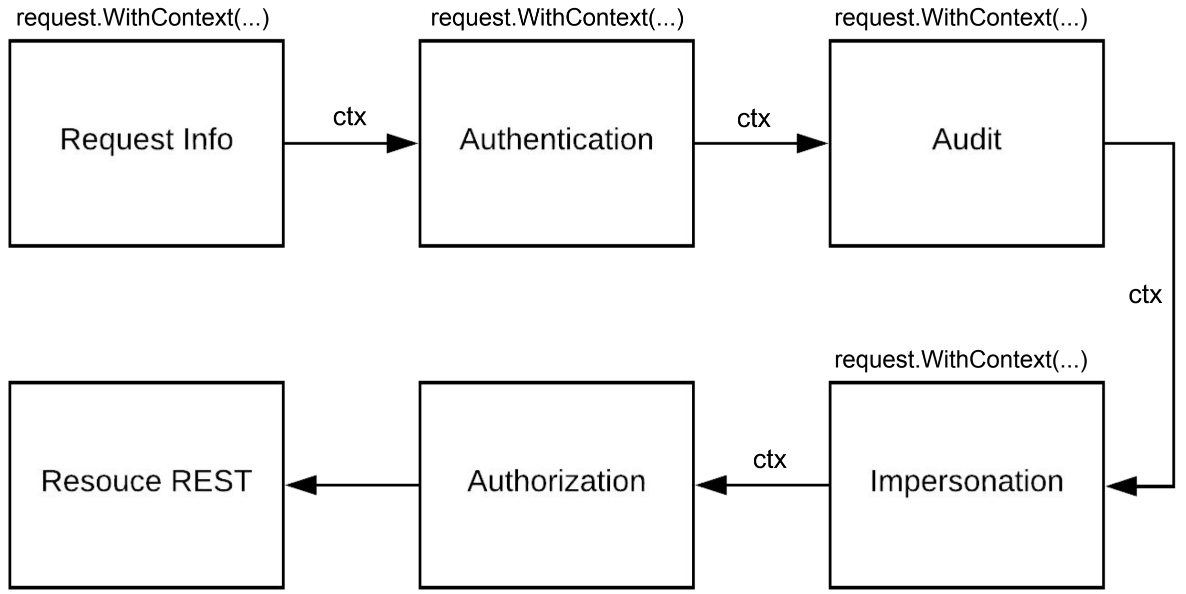
Registering the handler stack

```
authorizer := ...
authenticator := ...

handler := businessLogic()
handler = WithAuthorization(handler, authorizer)
handler = WithAuthentication(handler, authenticator)
```



Kubernetes request flow (simplified)



Adding to context

```
oldContext := req.Context()
req = req.WithContext(
    ctx.WithValue(oldContext, key, val),
)
```

Context Interface

```
type Context interface {
  ...
  Value(key interface{}) interface{}
}



func WithValue(parent Context, key, val interface{}) Context {
    ...
}
```

Context Implementation

```go
type valueCtx struct {
    Context
    key, val interface{}
}

func (c *valueCtx) Value(key interface{}) interface{} {
    if c.key == key {
        return c.val
    }
    return c.Context.Value(key)
}
```

Unique Context Key

```
type myKeyType int

const myUniqueKey myKeyType = iota
```

Context

```
myData := &MyDataType{}
ctx.WithValue(parentContext, myUniqueKey, myData)
```

Context

```
myData, ok := context.Value(myUniqueKey).(*MyDataType)
```

Context

```
oldContext := req.Context()
req = req.WithContext(
    ctx.WithValue(oldContext, myUniqueKey, myData),
)

---------------------------------------------------------

myData, ok := req.Context().Value(myUniqueKey).(*MyDataType)
```

```
(*context.valueCtx)({
    Context: (*context.valueCtx)({
        Context: (*context.cancelCtx)({
            Context: (*context.valueCtx)({
                ...
                key: (request.requestInfoKeyType) 0,
                val: (*request.RequestInfo)({...})
            }),
            ...
        }),
        key: (request.key) 1,
        val: (*user.DefaultInfo)({
            Name: "system:anonymous",
            UID: "",
            Groups: { ["system:unauthenticated"] },
            Extra: (map[string][]string) {}
        })
    }),
    key: (request.key) 2,
    val: (*audit.Event)({
        ...
        Level: (audit.Level) "Metadata",
        AuditID: (types.UID) "5762f2ad-44f1-4c7a-a4dd-56be280b0841",
        ...
    })
})
```

request

```
RequestInfo {
     IsResourceRequest: true,
     Path: “/api/v1/namespaces/kube-system/pods/etcd-quorum-guard/log”
     Verb: “get”,
     APIPrefix: “api”,
     APIGroup: “”,
     APIVersion: “v1”,
     Namespace: “kube-system”,
     Resource: “pods”,
     Subresource: “log”,
     Name: “etcd-quorum-guard”,
     Parts: {
          “pods”,
          “etcd-quorum-guard”,
          “log”,
     }
}
```

An API resource, not URL

API request type (not http type)

Resource type

k8s.io/apiserver/pkg/server/config.go:

```
func DefaultBuildHandlerChain(apiHandler http.Handler, c *Config) http.Handler {
	handler := genericapifilters.WithAuthorization(apiHandler, c.Authorization.Authorizer, ...)
	handler = genericfilters.WithMaxInFlightLimit(handler, c.MaxRequestsInFlight, ...
	handler = genericapifilters.WithImpersonation(handler, c.Authorization.Authorizer, ...)
	handler = genericapifilters.WithAudit(handler, c.AuditBackend, ...)
	failedHandler := genericapifilters.Unauthorized(c.Serializer, c.Authentication.SupportsBasicAuth)
	failedHandler = genericapifilters.WithFailedAuthenticationAudit(failedHandler, c.AuditBackend, ..
	handler = genericapifilters.WithAuthentication(handler, c.Authentication.Authenticator, ...)
	handler = genericfilters.WithCORS(handler, c.CorsAllowedOriginList, ...)
	handler = genericfilters.WithTimeoutForNonLongRunningRequests(handler, c.LongRunningFunc, ...)
	handler = genericfilters.WithWaitGroup(handler, c.LongRunningFunc, c.HandlerChainWaitGroup)
	handler = genericapifilters.WithRequestInfo(handler, c.RequestInfoResolver)
	handler = genericfilters.WithPanicRecovery(handler)
	return handler
}
```

k8s.io/apiserver/pkg/endpoints/filters/authentication.go

```
func WithAuthentication(handler http.Handler, auth authenticator.Request, ...) http.Handler {
	if auth == nil {
		return handler
	}


	return http.HandlerFunc(func(w http.ResponseWriter, req *http.Request) {
		user, ok, err := auth.AuthenticateRequest(req)
		if err != nil || !ok {
			...
			return
		}

		req = req.WithContext(
			WithUser(req.Context(), user),
		)

		handler.ServeHTTP(w, req)
	})
}
```

k8s.io/apiserver/pkg/authentication/authenticator/interfaces.go:

```
type Request interface {
    AuthenticateRequest(req *http.Request) (*Response, bool, error)
}
```

```
k8s.io/apiserver/pkg/authentication/authenticator/interfaces.go:

type Response struct {
    Audiences Audiences
    User user.Info
}

type Audiences []string
```

```
k8s.io/apiserver/pkg/authentication/user/user.go:


type Info interface {

    GetName() string

    GetUID() string

    GetGroups() []string

    GetExtra() map[string][]string
}
```

k8s.io/apiserver/pkg/authentication/request/x509/x509.go:

```
func (a *Authenticator) AuthenticateRequest(req *http.Request) (*authenticator.Response, bool, error) {
	...
	chains, err := req.TLS.PeerCertificates[0].Verify(optsCopy)
	if err != nil {
		return nil, false, err
	}

	var errlist []error
	for _, chain := range chains {
		user, ok, err := a.user.User(chain)
		if err != nil {
			errlist = append(errlist, err)
			continue
		}

		if ok {
			return user, ok, err
		}
	}
	return nil, false, utilerrors.NewAggregate(errlist)
}
```

```
type VerifyOptions struct {
	Roots        *CertPool
	KeyUsages []ExtKeyUsage
	...
}
```

```
Roots = CA bundle
KeyUsages = ExtKeyUsageClientAuth
```

```
k8s.io/apiserver/pkg/authentication/request/x509/x509.go:
```

```
var CommonNameUserConversion = UserConversionFunc(func(chain []*x509.Certificate) (*authenticator.Response, bool, error) {
	if len(chain[0].Subject.CommonName) == 0 {
		return nil, false, nil
	}
	return &authenticator.Response{
		User: &user.DefaultInfo{
			Name:   chain[0].Subject.CommonName,
			Groups: chain[0].Subject.Organization,
		},
	}, true, nil
})
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: OU = bootkube, CN = kube-ca
        Validity
            Not Before: Feb 11 19:09:20 2019 GMT
            Not After : Feb  8 19:09:20 2029 GMT
        Subject: OU = foo, CN = bar                         ==      &user.DefaultInfo{Name: “bar”, Groups: “foo”}
```

```
k8s.io/apiserver/pkg/authentication/request/headerrequest/requestheader.go

func (a *requestHeaderAuthRequestHandler) AuthenticateRequest(req *http.Request) (*authenticator.Response, bool, error) {
	name := headerValue(req.Header, a.nameHeaders)
	if len(name) == 0 {
		return nil, false, nil
	}
	groups := allHeaderValues(req.Header, a.groupHeaders)
	extra := newExtra(req.Header, a.extraHeaderPrefixes)

	...

	return &authenticator.Response{
		User: &user.DefaultInfo{
			Name:   name,
			Groups: groups,
			Extra:  extra,
		},
	}, true, nil
}
```

```
k8s.io/apiserver/pkg/authentication/request/x509/x509.go:

func (a *Verifier) AuthenticateRequest(req *http.Request) (*authenticator.Response, bool, error) {
      ...

      if _, err := req.TLS.PeerCertificates[0].Verify(optsCopy); err != nil {
            return nil, false, err
      }

      if err := a.verifySubject(req.TLS.PeerCertificates[0].Subject); err != nil {
            return nil, false, err
      }

      return a.auth.AuthenticateRequest(req)
}
```

Optional common name verification

The wrapped request header authenticator

```
k8s.io/apiserver/pkg/authentication/authenticator/interfaces.go:


type Token interface {
     AuthenticateToken(ctx context.Context, token string) (*Response, bool, error)
}
```

```
k8s.io/apiserver/pkg/authentication/request/bearertoken/bearertoken.go

func (a *Authenticator) AuthenticateRequest(req *http.Request) (*authenticator.Response, bool, error)
{
	auth := strings.TrimSpace(req.Header.Get("Authorization"))

	...

	parts := strings.Split(auth, " ")

	...

	token := parts[1]

	...

	resp, ok, err := a.auth.AuthenticateToken(req.Context(), token)

	...

	return resp, ok, err
}
```

k8s.io/apiserver/plugin/pkg/authenticator/token/oidc/oidc.go

```go
func (a *Authenticator) AuthenticateToken(ctx context.Context, token string) (*authenticator.Response, bool, error) {
    if reqAuds, ok := authenticator.AudiencesFrom(ctx); ok {
        if len(reqAuds.Intersect(a.clientIDs)) == 0 && len(reqAuds.Intersect(a.apiAudiences)) == 0 {
                return nil, false, nil
        }
    }
    if !hasCorrectIssuer(a.issuerURL, token) {
        return nil, false, nil
    }

    ...

    idToken, err := verifier.Verify(ctx, token)
    if err != nil { ... }

    ...

    var username string
    if err := c.unmarshalClaim(a.usernameClaim, &username); err != nil { ... }

    ...

    info := &user.DefaultInfo{Name: username}

    ...

    var groups stringOrArray
    if err := c.unmarshalClaim(a.groupsClaim, &groups); err != nil { ... }
    info.Groups = []string(groups)

    ...

    return &authenticator.Response{User: info}, true, nil
}
```

```go
type Options struct {
    IssuerURL string

    CAFile string

    UsernameClaim string
    GroupsClaim string

    ...
}
```

k8s.io/kubernetes/pkg/serviceaccount/jwt.go

```go
func (j *jwtTokenAuthenticator) AuthenticateToken(ctx context.Context, tokenData string) (*authenticator.Response, bool, error) {
	if !j.hasCorrectIssuer(tokenData) {
		return nil, false, nil
	}

	tok, err := jwt.ParseSigned(tokenData)
	if err != nil {
		return nil, false, nil
	}

	public := &jwt.Claims{}
	private := j.validator.NewPrivateClaims()
	...
		<<process claims and audiences>>
	...
	sa, err := j.validator.Validate(tokenData, public, private)
	if err != nil {
		return nil, false, err
	}

	return &authenticator.Response{
		User:      sa.UserInfo(),
		Audiences: auds,
	}, true, nil
}
```

k8s.io/apiserver/pkg/authentication/token/cache/cached_token_authenticator.go

```go
func (a *cachedTokenAuthenticator) AuthenticateToken(ctx context.Context, token string)
(*authenticator.Response, bool, error) {
      auds, _ := authenticator.AudiencesFrom(ctx)

      key := keyFunc(auds, token)
      if record, ok := a.cache.get(key); ok {
            return record.resp, record.ok, record.err
      }

      resp, ok, err := a.authenticator.AuthenticateToken(ctx, token)
      if !a.cacheErrs && err != nil {
            return resp, ok, err
      }

      switch {
      case ok && a.successTTL > 0:
            a.cache.set(key, &cacheRecord{resp: resp, ok: ok, err: err}, a.successTTL)
      case !ok && a.failureTTL > 0:
            a.cache.set(key, &cacheRecord{resp: resp, ok: ok, err: err}, a.failureTTL)
      }

      return resp, ok, err
}
```

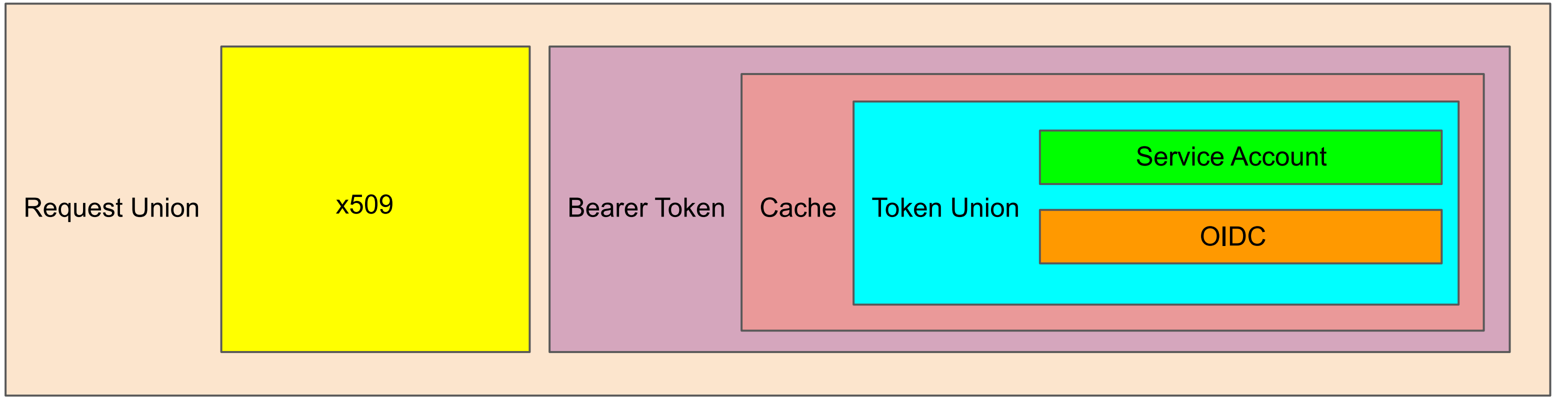
k8s.io/apiserver/pkg/authentication/request/union/union.go
k8s.io/apiserver/pkg/authentication/token/union/union.go

```go
type unionAuthRequestHandler struct {
      Handlers []authenticator.Request
      FailOnError bool
}

func (a *unionAuthRequestHandler) AuthenticateRequest(req *http.Request) (*authenticator.Response, bool, error) {
    var errlist []error
    for _, currAuthRequestHandler := range authHandler.Handlers {
       resp, ok, err := currAuthRequestHandler.AuthenticateRequest(req)
       if err != nil {
             if a.FailOnError {
                   return resp, ok, err
             }
             errlist = append(errlist, err)
             continue
       }

       if ok {
             return resp, ok, err
       }
    }

    return nil, false, utilerrors.NewAggregate(errlist)
}
```


Request Union
x509
Bearer Token
Cache
Token Union
Service Account
OIDC

k8s.io/apiserver/pkg/authentication/authenticatorfactory/delegating.go:

```
func (c DelegatingAuthenticatorConfig) New() (authenticator.Request, ..., error) {
	authenticators := []authenticator.Request{}
	...
	if c.RequestHeaderConfig != nil {
		requestHeaderAuthenticator, err := headerrequest.NewSecure(...)
		...
		authenticators = append(authenticators, requestHeaderAuthenticator)
	}
	...
	if len(c.ClientCAFile) > 0 {
		...
		authenticators = append(authenticators, x509.New(verifyOpts, x509.CommonNameUserConversion))
	}
	if c.TokenAccessReviewClient != nil {
		...
		cachingTokenAuth := cache.New(tokenAuth, ...)
		authenticators = append(authenticators, bearertoken.New(cachingTokenAuth), ...)
		...
	}
	...
	authenticator := group.NewAuthenticatedGroupAdder(unionauth.New(authenticators...))
	...
	return authenticator, ..., nil
}
```

k8s.io/apiserver/pkg/endpoints/filters/authorization.go

```go
func WithAuthorization(handler http.Handler, a authorizer.Authorizer, ...) http.Handler {
	if a == nil {
		return handler
	}

	return http.HandlerFunc(func(w http.ResponseWriter, req *http.Request) {
		…

		attributes, err := GetAuthorizerAttributes(ctx)

		…

		authorized, reason, err := a.Authorize(attributes)

		if authorized == authorizer.DecisionAllow {
			handler.ServeHTTP(w, req)
			return
		}
		if err != nil {
			...
			responsewriters.InternalError(w, req, err)
			return
		}

		…

		responsewriters.Forbidden(ctx, attributes, w, req, reason, s)
	})
}
```

k8s.io/apiserver/pkg/authorization/authorizer/interfaces.go

```
type Authorizer interface {
    Authorize(a Attributes) (authorized Decision, reason string, err error)
}



type Decision int

const (
    DecisionDeny                Decision = iota
    DecisionAllow
    DecisionNoOpinion
)



type Attributes interface {
    IsReadOnly() bool                      verb == get, list, watch

    User info
    Request info
}
```

k8s.io/apiserver/pkg/authorization/authorizerfactory/builtin.go

```
type privilegedGroupAuthorizer struct {
    groups []string                                        ==     system:masters
}

func (r *privilegedGroupAuthorizer) Authorize(attr authorizer.Attributes) (authorizer.Decision, string, error) {
    ...
    for _, attr_group := range attr.GetUser().GetGroups() {
       for _, priv_group := range r.groups {
             if priv_group == attr_group {
                   return authorizer.DecisionAllow, "", nil
             }
       }
    }
    return authorizer.DecisionNoOpinion, "", nil
}
```

k8s.io/apiserver/plugin/pkg/authorizer/webhook/webhook.go

```
func (w *WebhookAuthorizer) Authorize(attr authorizer.Attributes) (decision authorizer.Decision, reason string, err error) {
    r := &authorization.SubjectAccessReview{}

    (convert attr to SAR)

    key, err := json.Marshal(r.Spec)
    ...
    if entry, ok := w.responseCache.Get(string(key)); ok {
        r.Status = entry.(authorization.SubjectAccessReviewStatus)
    } else {
        ...
        webhook.WithExponentialBackoff(w.initialBackoff, func() error {
                result, err = w.subjectAccessReview.Create(r)
                return err
        })
        ...
        r.Status = result.Status
        if shouldCache(attr) {
                ...
        }
    }
    switch {
    ...
    case r.Status.Denied:
        return authorizer.DecisionDeny, r.Status.Reason, nil
    case r.Status.Allowed:
        return authorizer.DecisionAllow, r.Status.Reason, nil
    default:
        return authorizer.DecisionNoOpinion, r.Status.Reason, nil
    }

}
```

```
type SubjectAccessReviewSpec struct {
   ResourceAttributes *ResourceAttributes
   User string
   ...
}

type ResourceAttributes struct {
   Namespace string
   Verb string
   Group string
   Resource string
   ...
}
```

RBAC - Role Based Access Control

Effective RBAC - Jordan Liggitt

https://www.youtube.com/watch?v=Nw1ymxcLIDI

k8s.io/kubernetes/plugin/pkg/auth/authorizer/rbac/rbac.go

```
k8s.io/apiserver/pkg/authorization/union/union.go

type unionAuthzHandler []authorizer.Authorizer

func (authzHandler unionAuthzHandler) Authorize(a authorizer.Attributes) (authorizer.Decision, string, error) {
    var (
       errlist    []error
       reasonlist []string
    )

    for _, currAuthzHandler := range authzHandler {
       decision, reason, err := currAuthzHandler.Authorize(a)

       if err != nil {
             errlist = append(errlist, err)
       }
       if len(reason) != 0 {
             reasonlist = append(reasonlist, reason)
       }
       switch decision {
       case authorizer.DecisionAllow, authorizer.DecisionDeny:
             return decision, reason, err
       case authorizer.DecisionNoOpinion:
             // continue to the next authorizer
       }
    }

    return authorizer.DecisionNoOpinion, strings.Join(reasonlist, "\n"), utilerrors.NewAggregate(errlist)
}
```

k8s.io/apiserver/pkg/server/options/authorization.go

```
func (s *DelegatingAuthorizationOptions) toAuthorizer(client kubernetes.Interface) (authorizer.Authorizer, error) {
    var authorizers []authorizer.Authorizer

    if len(s.AlwaysAllowGroups) > 0 {
       authorizers = append(authorizers, authorizerfactory.NewPrivilegedGroups(s.AlwaysAllowGroups...))
    }

    if len(s.AlwaysAllowPaths) > 0 {
       a, err := path.NewAuthorizer(s.AlwaysAllowPaths)
       ...
       authorizers = append(authorizers, a)
    }

    ...
    cfg := authorizerfactory.DelegatingAuthorizerConfig{
       SubjectAccessReviewClient: client.AuthorizationV1beta1().SubjectAccessReviews(),
       ...
    }
    delegatedAuthorizer, err := cfg.New()
    ...
    authorizers = append(authorizers, delegatedAuthorizer)

    return union.New(authorizers...), nil
}
```

```go
type Info interface {
    ...

    // GetExtra can contain any additional information that the authenticator
    // thought was interesting.  One example would be scopes on a token.
    // Keys in this map should be namespaced to the authenticator or
    // authenticator/authorizer pair making use of them.
    // For instance: "example.org/foo" instead of "foo"

    // This is a map[string][]string because it needs to be serializeable into
    // a SubjectAccessReviewSpec.authorization.k8s.io for proper authorization
    // delegation flows

    GetExtra() map[string][]string
}
```

Example: Scopes in OpenShift

Key

scopes.authorization.openshift.io

Values

1. user:info
2. user:check-access

github.com/openshift/origin/pkg/authorization/authorizer/scope/converter.go

```
func (a *scopeAuthorizer) Authorize(attributes authorizer.Attributes) (authorizer.Decision, string, error) {
    ...

    scopes := user.GetExtra()[authorizationapi.ScopesKey]
    if len(scopes) == 0 {
       return authorizer.DecisionNoOpinion, "", nil
    }

    ...

    rules, err := ScopesToRules(scopes, attributes.GetNamespace(), a.clusterRoleGetter)

    ...

    if authorizerrbac.RulesAllow(attributes, rules...) {
       return authorizer.DecisionNoOpinion, "", nil
    }

    return authorizer.DecisionDeny, ..., nil
}
```

```
type PolicyRule struct {
    Verbs []string
    APIGroups []string
    Resources []string
    ...
}
```

Ordering of Authorizers

What is wrong with this code?

```
Response{
    User: DefaultInfo{
       Name: "kube:admin",
       Groups: []string{"system:masters"},
       Extra: map[string][]string{
             ScopesKey: []string{"user:info"},
       },
    },
}
```

```
Authorizers:

1.   system:masters
2.   scopes
3.   RBAC
```

Thanks for attending!

Slack channel: #sig-auth

Home page: https://github.com/kubernetes/community/tree/master/sig-auth

Mailing list: https://groups.google.com/forum/#!forum/kubernetes-sig-auth

Bi-weekly meetings Wednesday at 20:00 CET