

Towards an Attribute-Based Role-Based Access Control System

October 24, 2019

ApacheCon, Berlin

The RBAC Standard

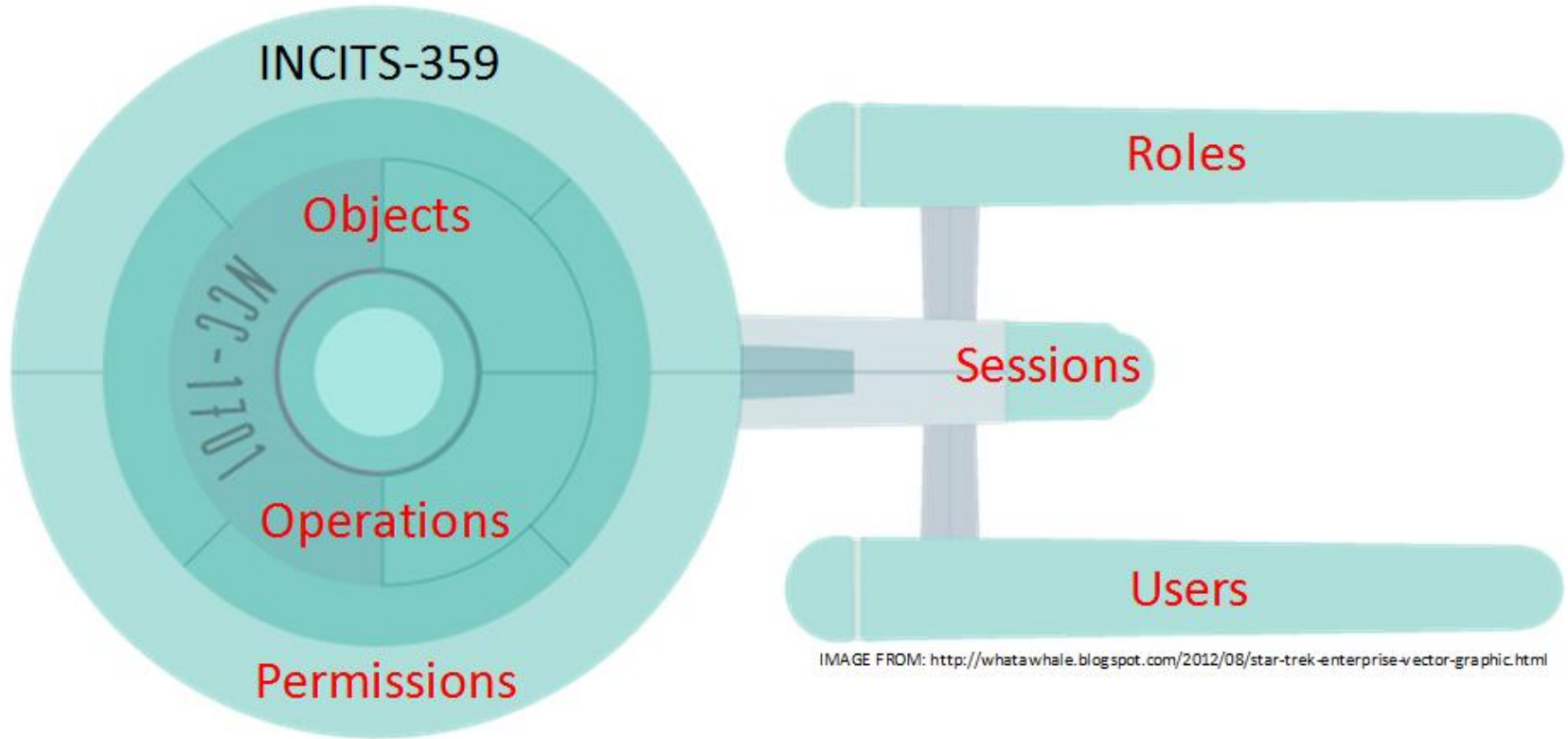


IMAGE FROM: <http://whatawhale.blogspot.com/2012/08/star-trek-enterprise-vector-graphic.html>

ANSI RBAC INCITS 359 Specification

RBAC0:

- Users, Roles, Perms, Sessions

RBAC1:

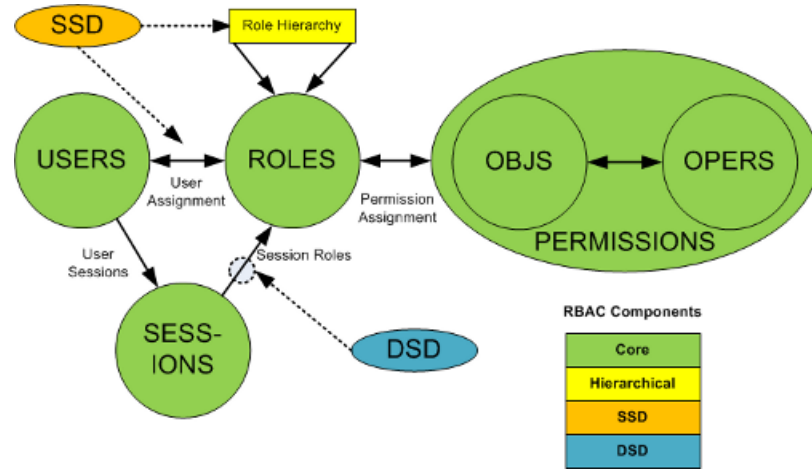
- Hierarchical Roles

RBAC2:

- Static Separation of Duties

RBAC3:

- Dynamic Separation of Duties



RBAC Object Model

Six basic elements:

1. **User** – human or machine entity
2. **Role** – a job function within an organization
3. **Object** – maps to system resources
4. **Operation** – executable image of program
5. **Permission** – approval to perform an Operation on one or more Objects
6. **Session** – contains set of activated roles for User

RBAC Functional Model

CreateSession(*user*, *session*)

This function creates a new session with a given user as owner and an active role set. The function is valid if and only if:

- the user is a member of the *USERS* data set, and
- the active role set is a subset of the roles assigned to that user. In a RBAC implementation, the session's active roles might actually be the groups that represent those roles.

The following schema formally describes the function. The *session* parameter, which represents the session identifier, is actually generated by the underlying system.

$CreateSession(user: NAME; ars: 2^{NAMES}; session: NAME) \triangleleft$

$user \in USERS; ars \subseteq \{r: ROLES \mid (user \mapsto r) \in UA\}; session \notin SESSIONS$

$SESSIONS' = SESSIONS \cup \{session\}$

$user_sessions' = user_sessions \setminus \{user \mapsto user_sessions(user)\} \cup$

$\{user \mapsto (user_sessions(user) \cup \{session\})\}$










$session_roles' = session_roles \cup \{session \mapsto ars\} \triangleright$

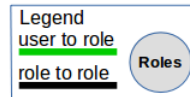
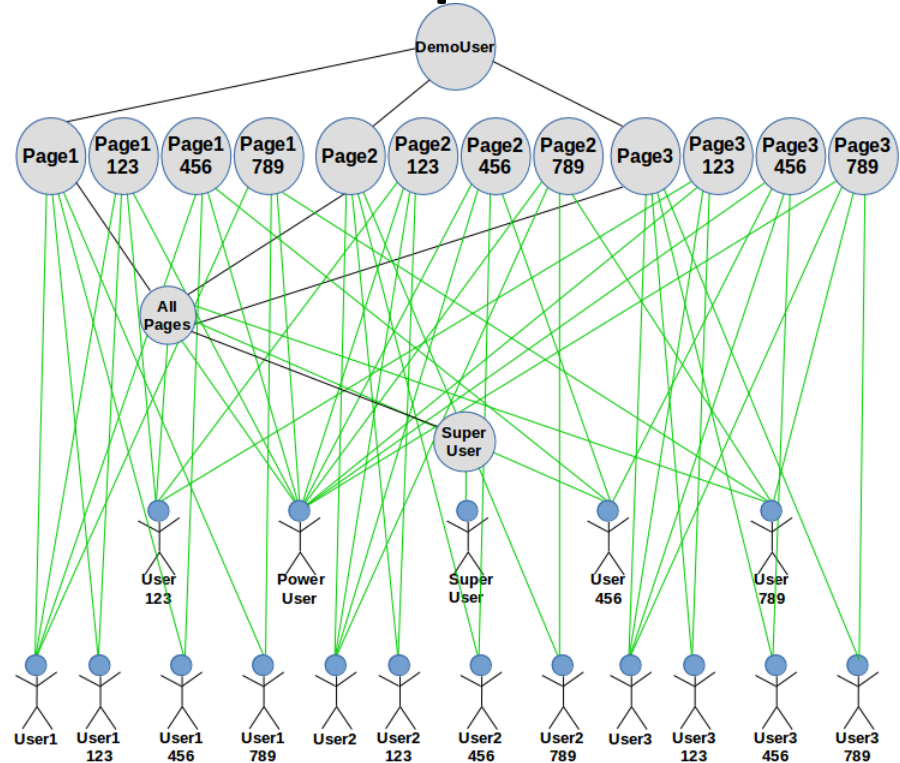
← Z-notation



Problem with Context: Role Explosion

▼  ou=Roles (17)

-  cn=PAGE1_123
-  cn=PAGE1_456
-  cn=PAGE1_789
-  cn=PAGE2_123
-  cn=PAGE2_456
-  cn=PAGE2_789
-  cn=PAGE3_123
-  cn=PAGE3_456
-  cn=PAGE3_789



Number of Roles = sizeof(A) * sizeof(B)

Roles (A)

Relationships (B)

Role1

Customer 123

Role2

*

Customer 456

=>

Role3

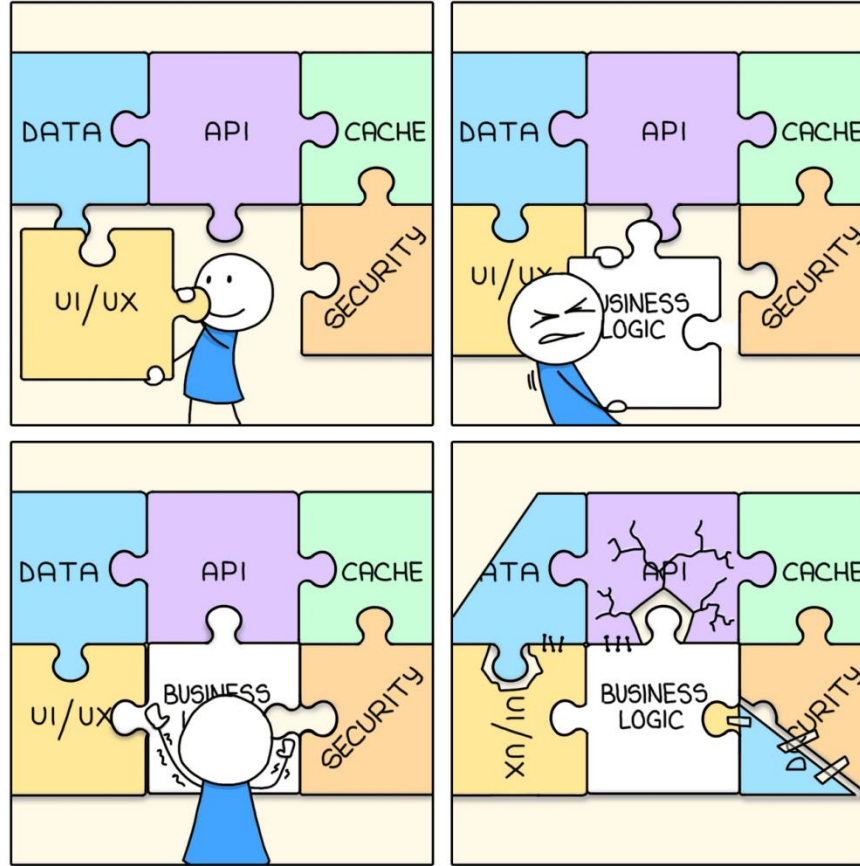
Customer 789

Roles

1. Role1-123
2. Role1-456
3. Role1-789
4. Role2-123
5. Role2-456
6. Role2-789
7. Role3-123
8. Role3-456
9. Role3-789

What now?

ARCHITECTURE



What is ABAC

An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions.

<https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-162.pdf>

Examples of ABAC

- Extensible Access Control Markup Language (XACML)
- Next Generation Access Control standard [ANSI499]

Enterprise ABAC

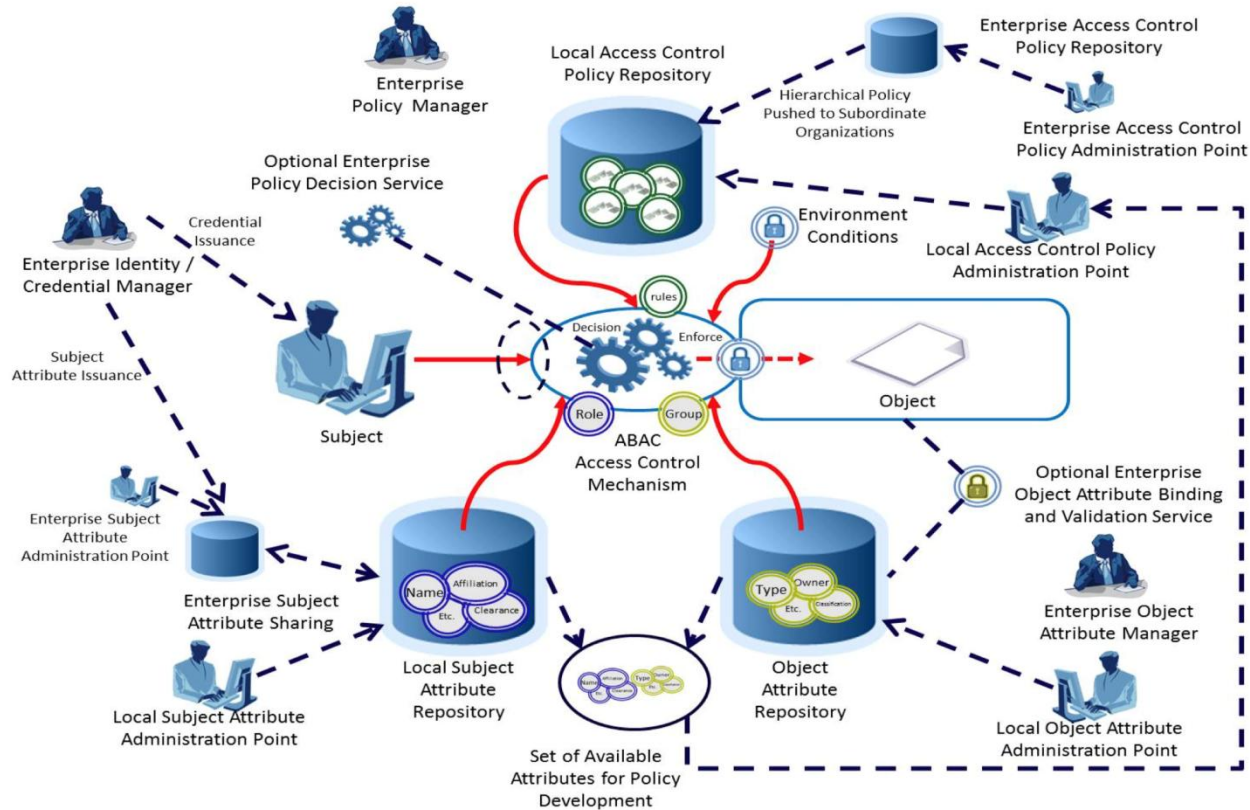


Figure 4: Enterprise ABAC Scenario Example

The Solution

Use attributes to constrain under what conditions roles may be activated.

Use Role Activation Phase

- Standard RBAC, Roles are assigned to Users.
- Roles must be activated into the Session.
- Principle of least privilege.

RBAC w/ ABAC

- Opportunity to introduce arbitrary attributes into the Role activation phase.
- The Role is 'special' in that it will only be activated if its conditions match.

Advantages

- Roles are no longer exploding.
- Continue to use RBAC, simpler to implement and maintain.
- No limit to the types of attributes.

e.g.

Roles:

- Teller
- Coin Washer

Constraints:

- Location

e.g. User-Role-Constraint

- Curly
 - Coin Washer: North
 - Coin Washer: South
 - Teller: East
- Moe
 - Coin Washer: East
 - Teller: North
 - Teller: South
- Larry
 - Coin Washer: West
 - Teller: West

Role Constraints

```
constraint role="Coin Washer"  
  key="location"
```

```
constraint role="Teller"  
  key="location"
```

User-Role Constraints

```
userId="Curly"  
  role="Teller"  
  key="location" value="East"
```

```
userId="Curly"  
  role="Coin Washer"  
  key="location" value="North"
```

```
userId="Curly"  
  role="Coin Washer"  
  key="location" value="South"
```

Under the Hood



<https://appdevcloudworkshop.github.io/images/introduction/image16.png>

RBAC w/ ABAC

LDAP - uid=curly,ou=People,dc=example,dc=com - slapd local - Apache Directory Studio

Help

cn=default,ou=Policies,dc=ari dc=example,dc=com **uid=curly,ou=People,dc=exam**

DN: uid=curly,ou=People,dc=example,dc=com

Attribute Description	Value
rtcSystem	FALSE
ftRC	washers\$type\$USER\$locale\$south\$
ftRC	washers\$type\$USER\$locale\$north\$
ftRC	tellers\$type\$USER\$locale\$east\$

Code Sample

```
// Nothing new here:
User user = new User("curly");

// This is new:
RoleConstraint constraint = new RoleConstraint( );

// In practice we're not gonna pass hard-coded key-values in here:
constraint.setKey( "location" );
constraint.setValue( "north" );

// This is just boilerplate goop:
List<RoleConstraint> constraints = new ArrayList();
constraints.add( constraint );

try
{
    // Create the RBAC session with ABAC constraint -- location=north, asserted:
    Session session = accessMgr.createSession( user, constraints );
    ...
}
```

<https://github.com/shawnmckinney/fortress-abac-demo/blob/master/src/main/java/com/mycompany/MyBasePage.java>

Closing Thoughts

1. Standards-based RBAC allows attributes into the mix.
 - *Fine-grained Authorization*

Apache Fortress

- <https://directory.apache.org/fortress/>
- Twitter: @apache_fortress



**APACHE
FORTRESS**

ApacheCon 2019, Berlin

Examples

1. <https://github.com/shawnmckinney/fortress-abac-demo>
2. <https://github.com/shawnmckinney/rbac-abac-sample>

Contact Info

Twitter: [@shawnmckinney](https://twitter.com/shawnmckinney)

Work: <http://symas.com>

Email: smckinney@apache.org

Blog: <https://iamfortress.net>

Project: <https://directory.apache.org/fortress>