Enterprise Access Control Patterns For REST and Web APIs

Francois Lascelles

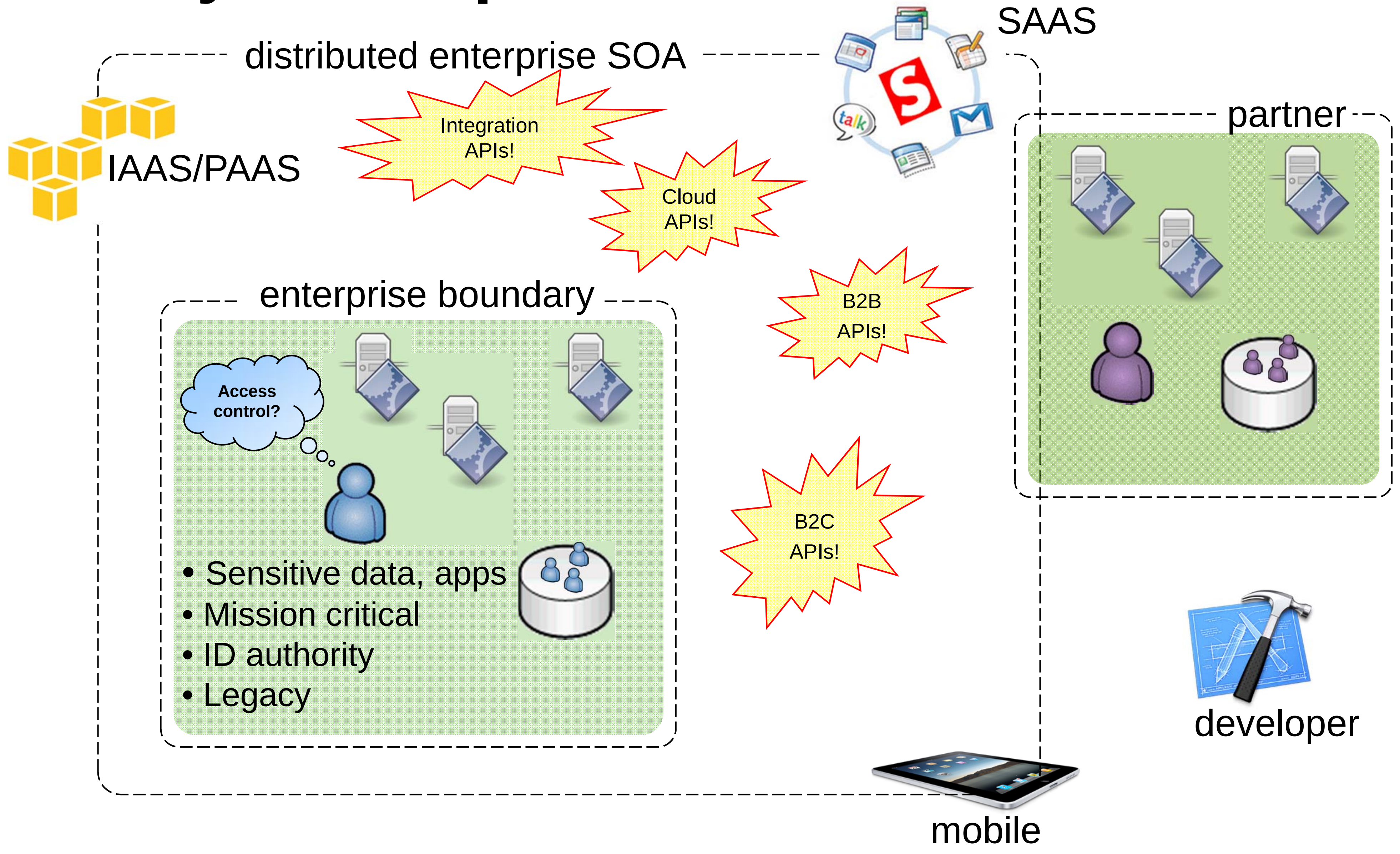
Layer 7 Technologies

Session ID: STAR-402

Session Classification: intermediate

RSA CONFERENCE 2012

Today's enterprise API drivers



- Sensitive data, apps
- Mission critical
- ID authority
- Legacy

REST access control standards gap

- WS-* web services have rich security standards and authentication/authorization mechanisms
- Web API, RESTful web services tend to use proprietary tokens, point-to-point solutions
- What are the common patterns in use?
- Which standards are emerging?
- How to use specialized infrastructure to implement access control?
- How to accommodate requesting party technical capabilities?

Pattern 1: API Keys in URI parameters

```
https://host/api/resource?keyid=foo&keysecret=bar
...
```

- Simplest thing, common practice
- Shared secret in a URL parameter based authentication, no signature involved
- Equivalent to https://host/api/resource?username=franco&password=mysecret
- Why not use HTTP Basic instead?

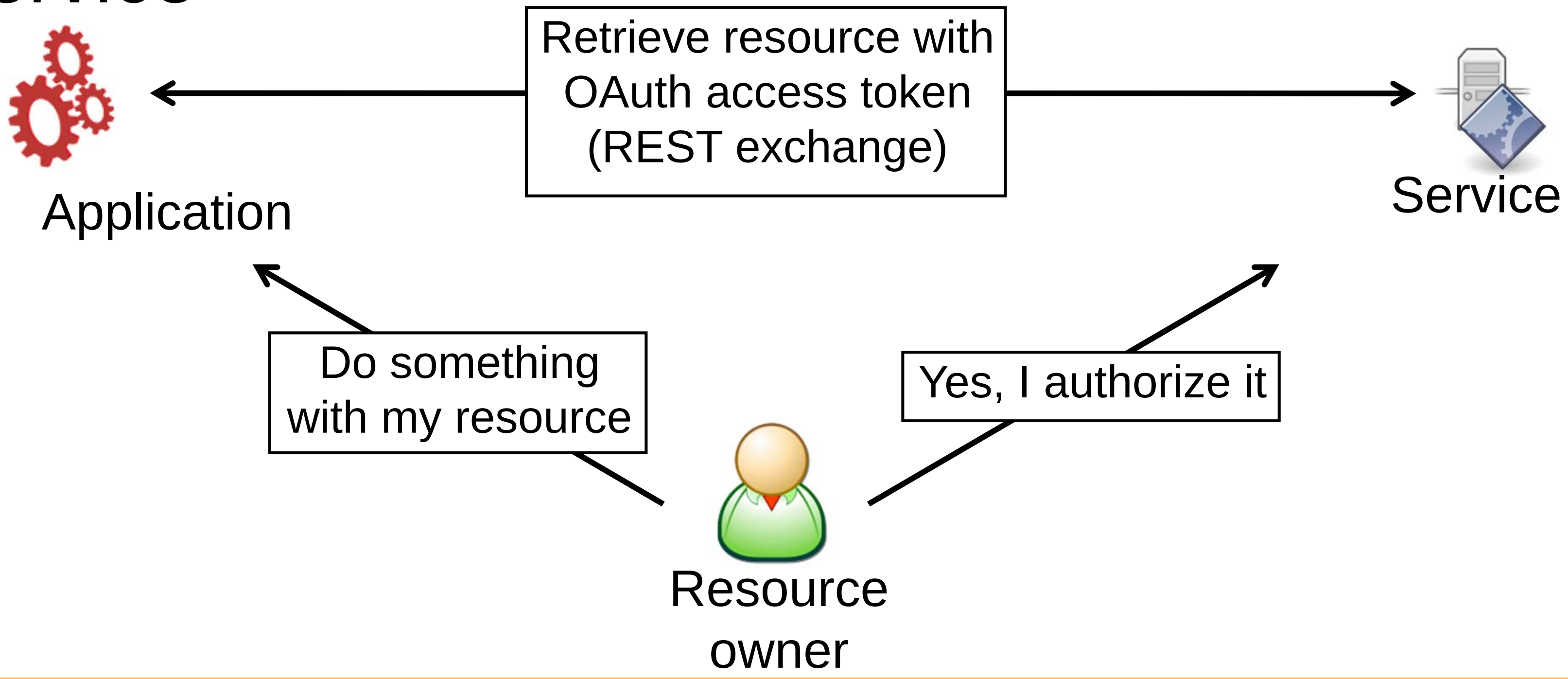
Pattern 2: HMAC

```
PUT /api/resource
…
Authorization: AWS keyid:fr0t5AzM6qT3S40pBPmfrTLJwMuZurA8=
…
```

- Prove possession of share secret using HMAC sig (shared secret not actually sent)
- Payload covered by signature -> message integrity
- Timestamp covered by signature -> less susceptible to replay
- Used by AWS, Azure, core to OAuth 1.0
- Requires agreement for normalized request string

Pattern 3: OAuth

- Specifies a handshake to grant an access token to an application (REST client)
- Access token is then used to consume REST service

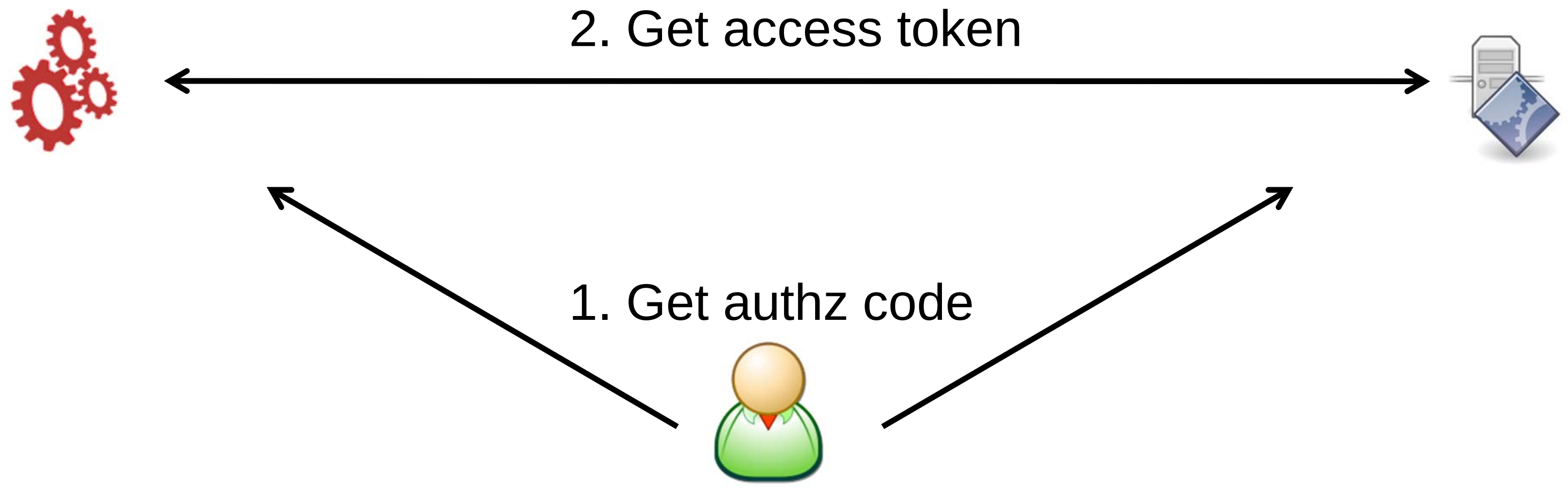

Retrieve resource with OAuth access token (REST exchange)
Application
Service
Do something with my resource
Yes, I authorize it
Resource owner

OAuth 2.0

- 4 core grant types (handshakes) to address different use cases
  - Authorization code, implicit, password, client credentials
- SAML extension grant type (draft-ietf-oauth-saml2-bearer-03)
- Different token types
  - Bearer (easy, like cookies)
  - MAC (integrity, more secure)
- OAuth 2.0 is rich, fills the standards gap

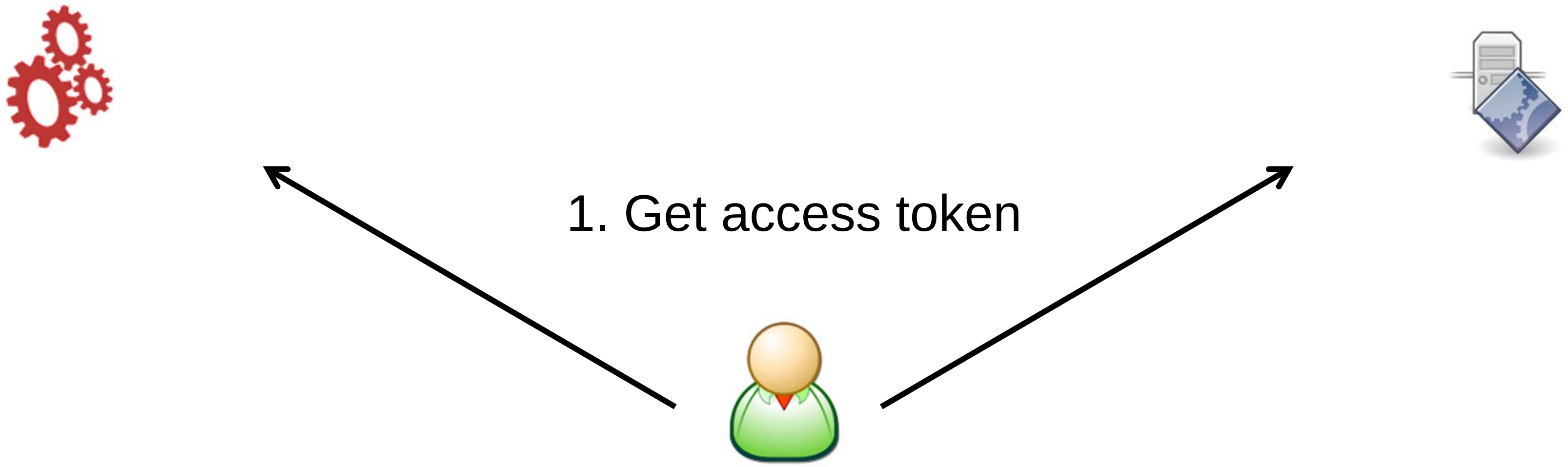
Authorization code grant type

- Resource owner redirected between OAuth authorization server and client application
- Both resource owner and client authenticated as part of handshake
- Supports refresh



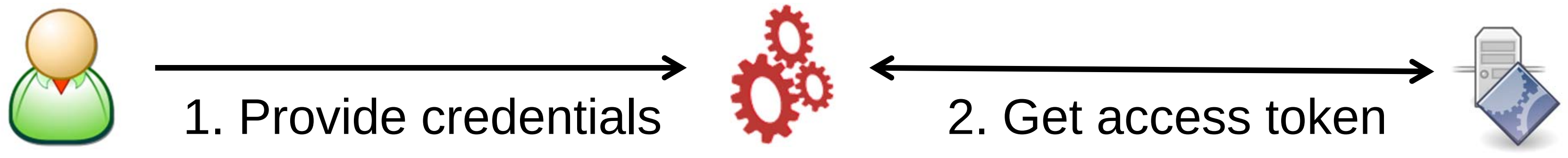
Implicit grant type

- Also 3-legged but simpler
- Client is not authenticated
  - redirection URI must be registered to avoid fishing
- No refresh



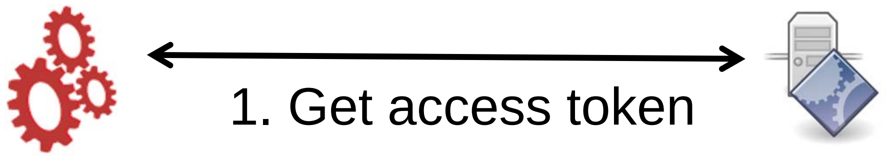
Resource owner password credentials grant type

- Resource owner provides credentials to client
- Client uses it to get access token
- Both client and res owner identities authenticated
- Can be refreshed

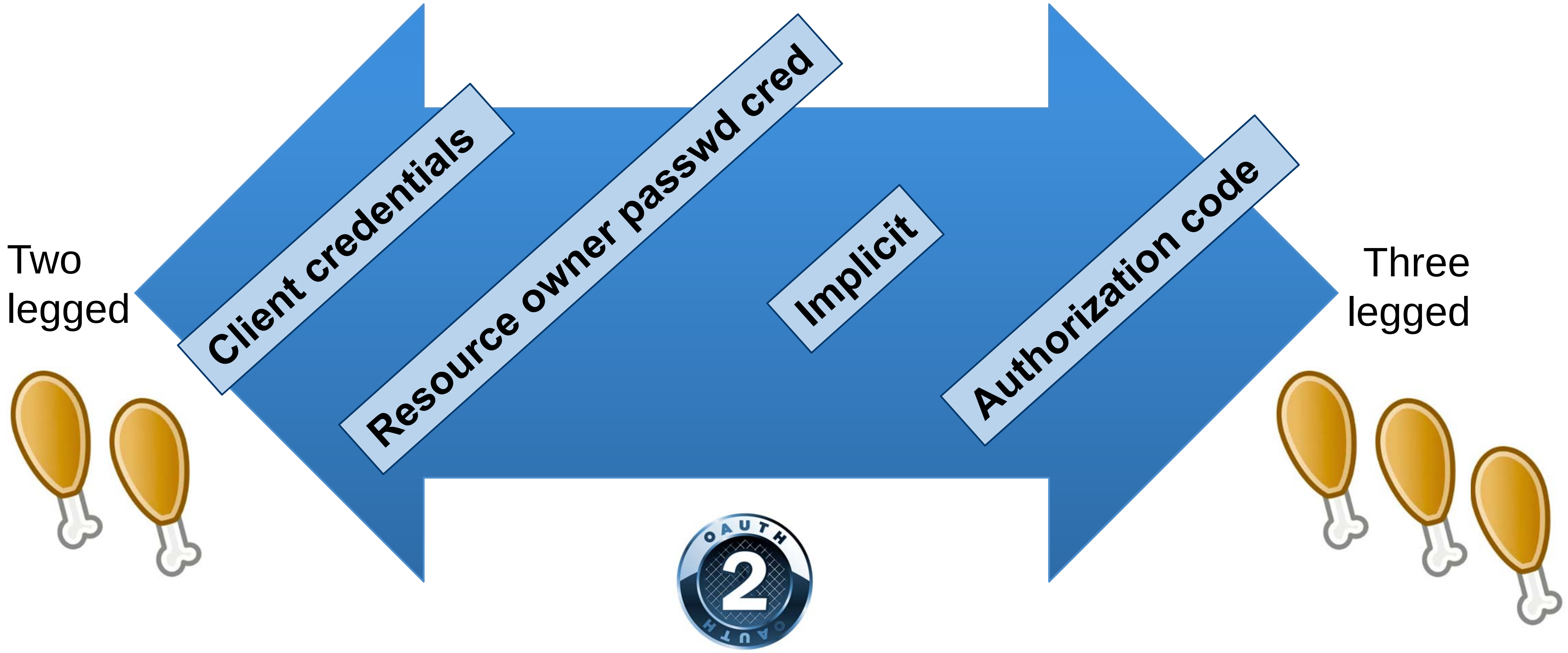


Client credentials grant type

- Two-legged handshake
- Client application authenticated only
- No refresh tokens



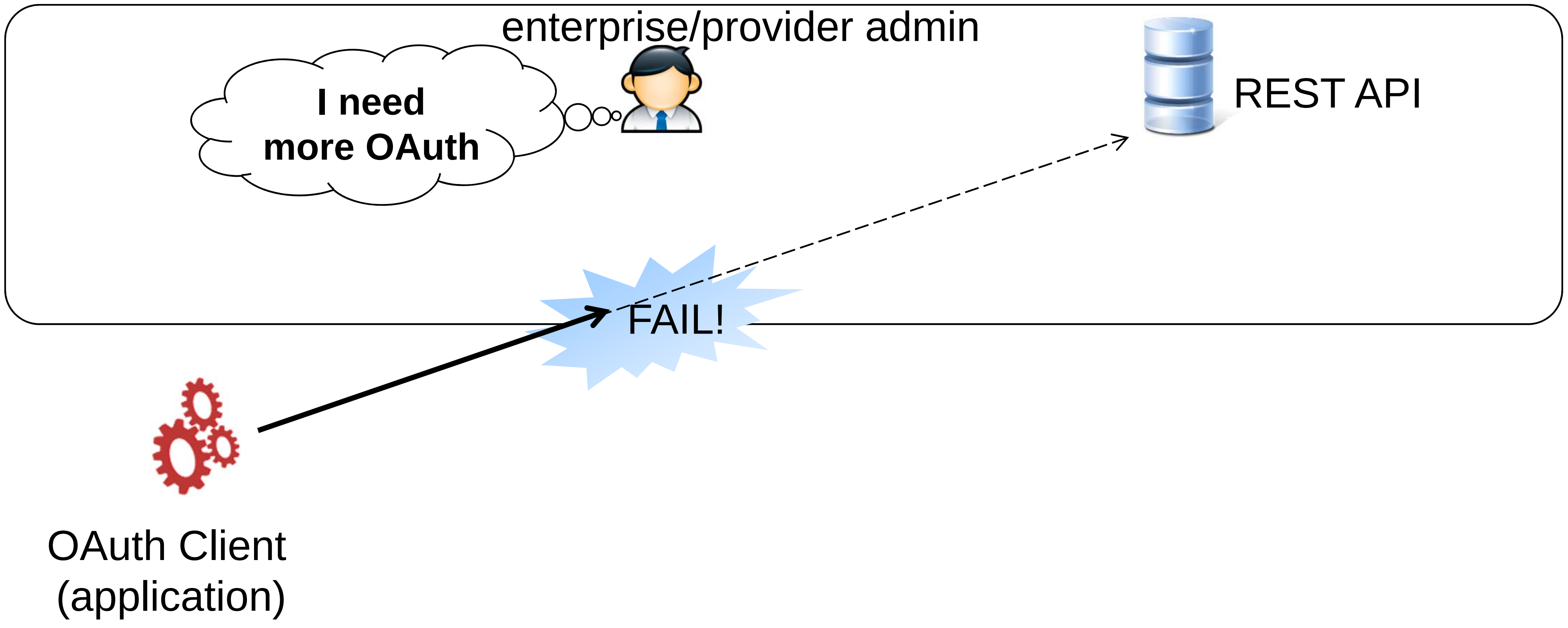
2 vs. 3 Legged Spectrum



Step-by-step enterprise API access control (from an OAuth perspective)

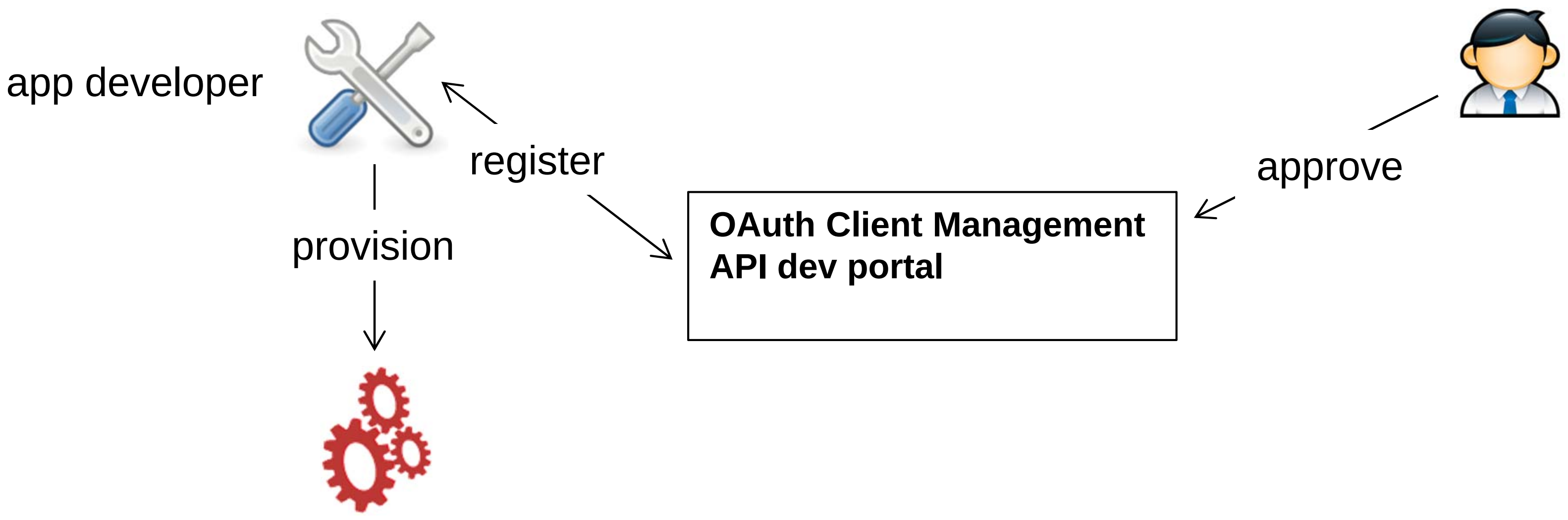
RSACONFERENCE2012

Starting Point



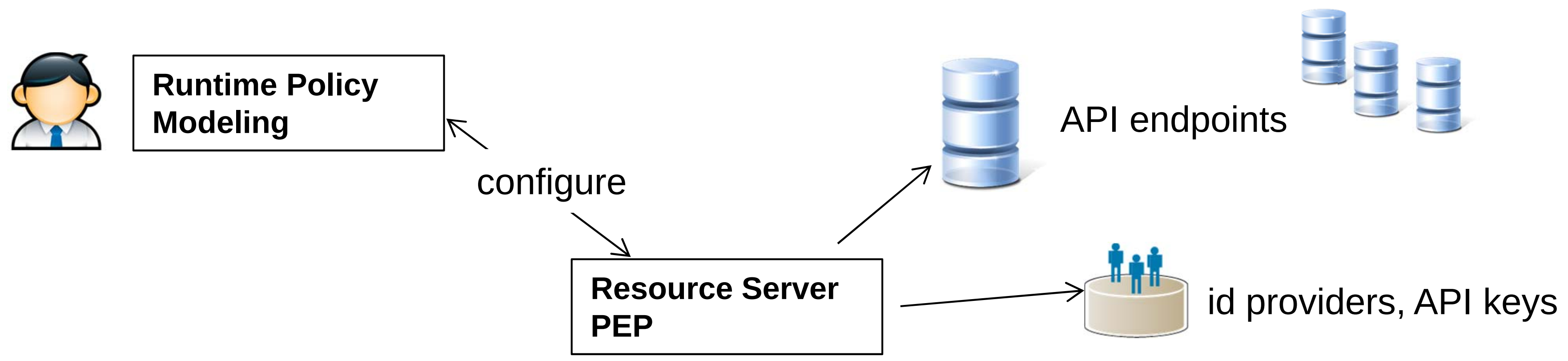
OAuth Clients Provisioning, Management

- Provide a portal for developers to register, generate shared secrets
- Enable approval flow (administrative)
- Store API keys, redirection URIs
- List existing clients, record usage statistics



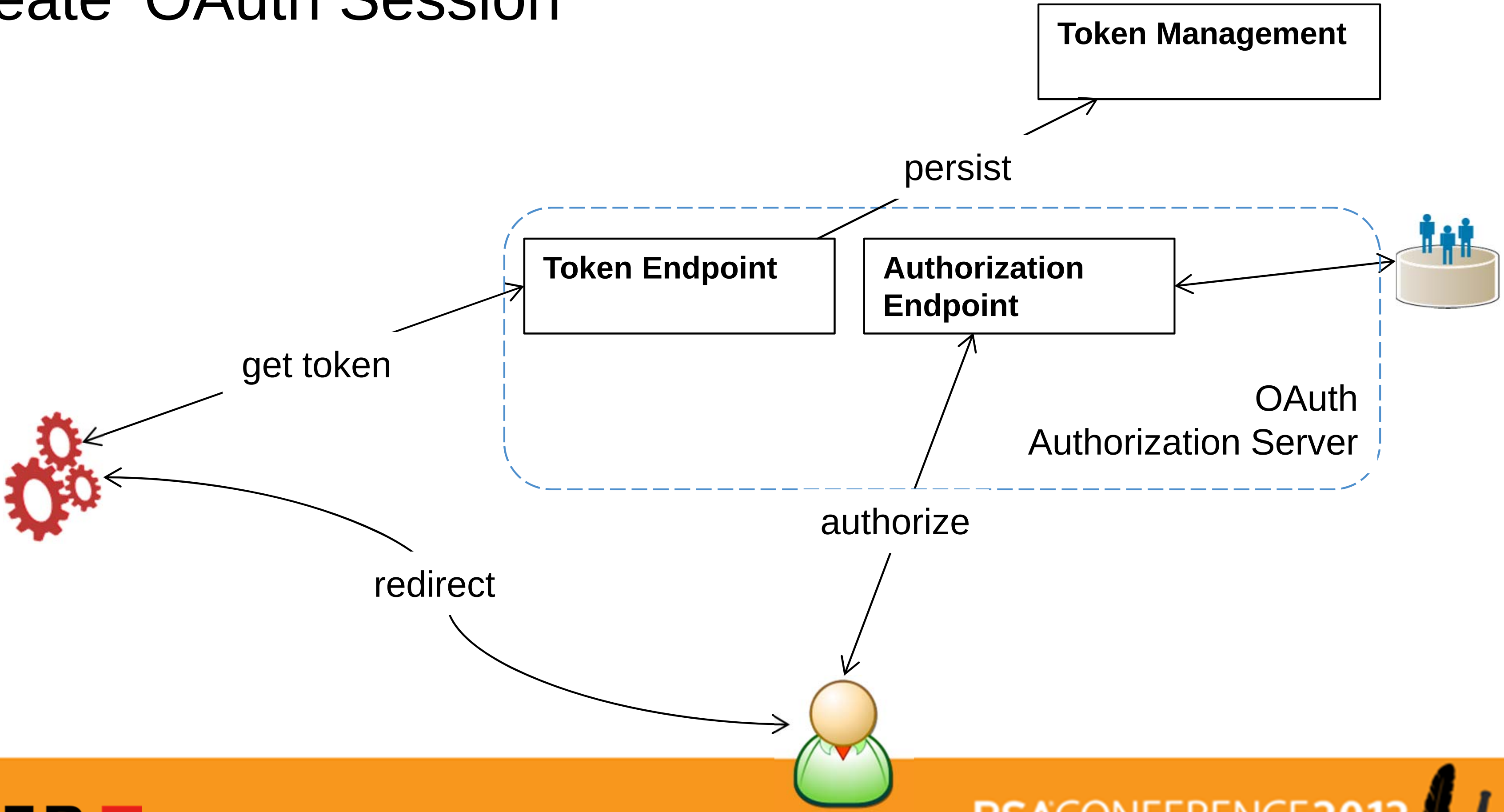
Runtime Policy Modeling, Integration

- Declare API endpoints in the resource server
- Integrate identity providers for runtime authentication
- Granular access control rules
  - Which API, which identities, which grant types, …



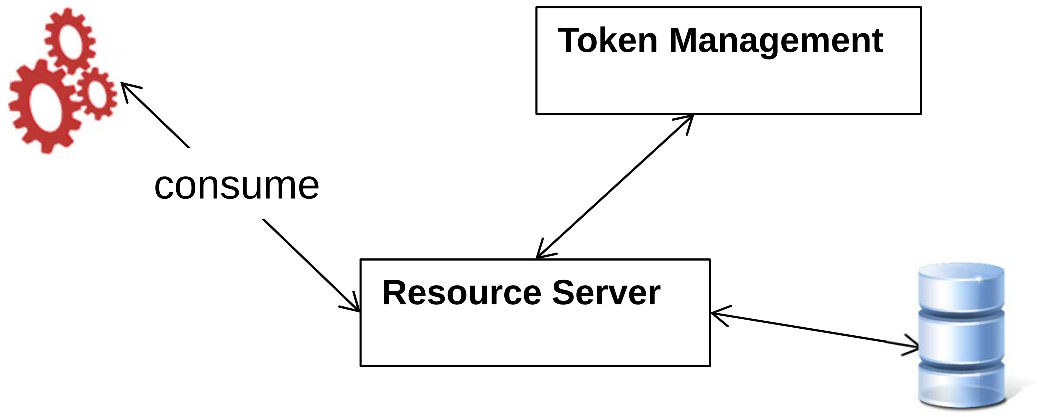
OAuth Handshake

- Enable handshake
  - Lookup policy, authenticate identities, enable flow
  - Create ‘OAuth Session’



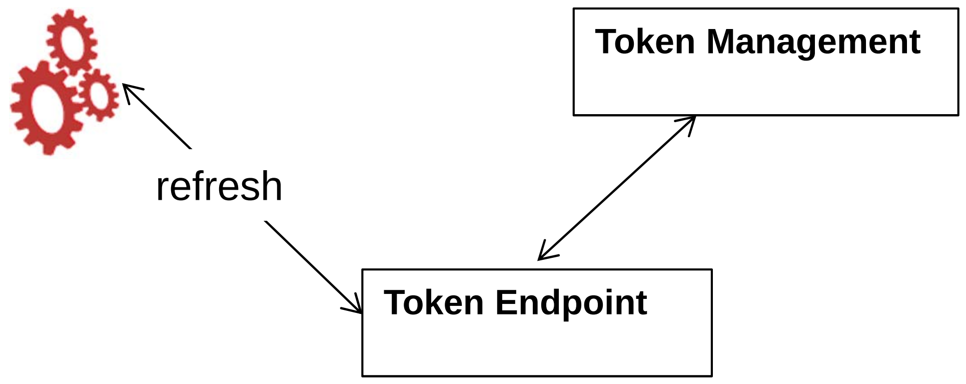
Runtime API Call

- OAuth resource server enables API call
  - Lookup and verify incoming OAuth access token
  - Authorize based on OAuth session attributes
  - Route to API endpoint, return resource to client app
  - Record consumption statistics



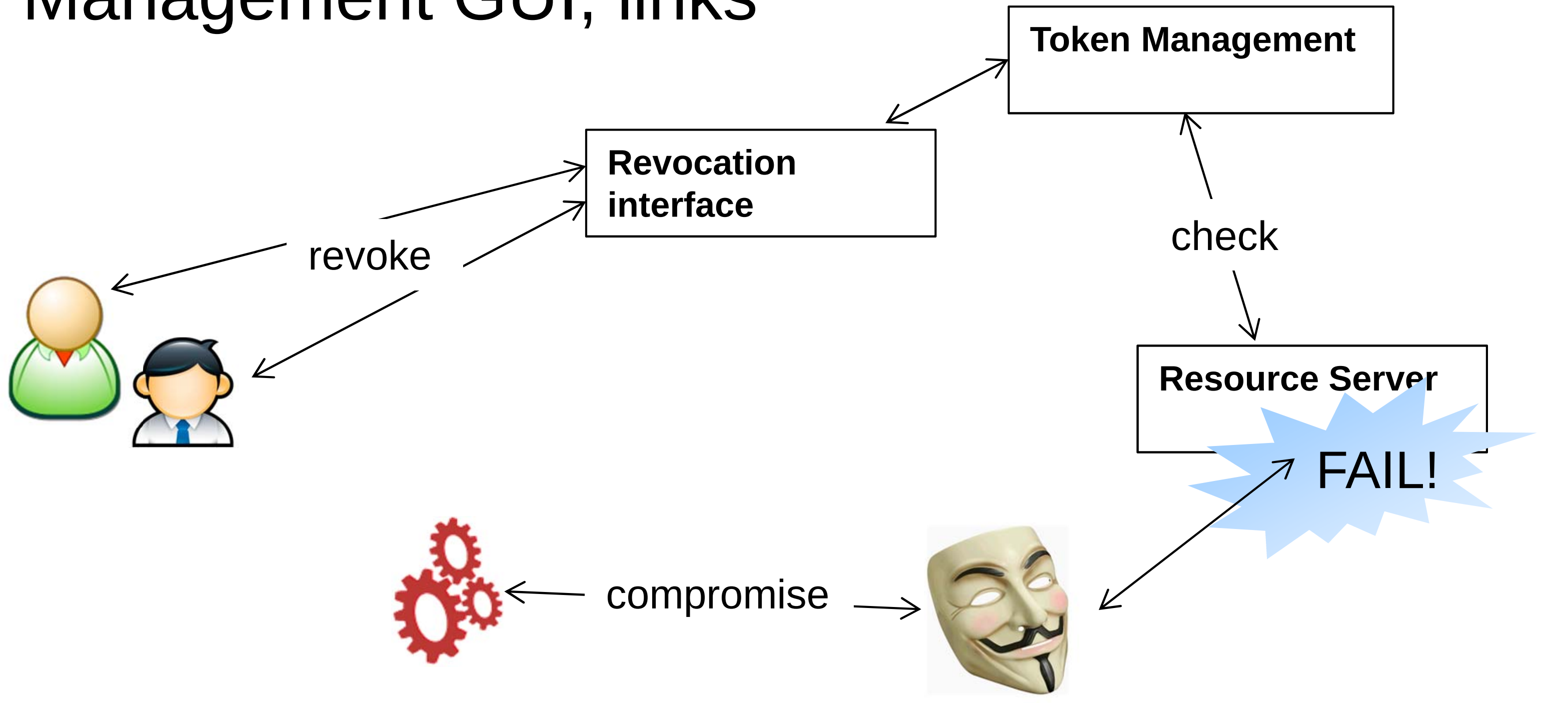
Token Refresh

- OAuth authorization server enables refresh
  - Authenticate client
  - Lookup and validate refresh token
  - Create new access token
  - Update ‘OAuth session’



Token Revocation

- Minimize impact of compromised tokens
- Enable revocation for subscribers and API providers
  - Management GUI, links



Comprehensive API Access Control

- Apply OAuth-enabling infrastructure:
  - Token management (lifecycle, revocation)
  - Developer portal (client provisioning, client management)
  - OAuth resource server (API proxy, PEP)
  - OAuth authorization server (authorization endpoint, token endpoint)
  - Runtime policy modeling
  - Reporting, monitoring interface

Thank you

For more information: info@layer7.com

RSA CONFERENCE 2012