apigee

Aug, 2015

Create and Manage APIs

Securing your API with API Keys

Protection against Content-Based Attacks

Use Regular Expressions to Block Access

Name	Regular Expression
SQL Injection	[\s]*((delete)|(exec)|(drop\s*table)|(insert)|(shutdown)|(update)|(\bor\b))
Server-Side Include Injection	<!--\s*<!--(include|exec|echo|config|printenv)\s+.*
XPath Abbreviated Syntax Injection	(/(@?[\w_?\w:*]+(\+\])*)?)+
XPath Expanded Syntax Injection	/?(ancestor(-or-self)?|descendant(-or-self)?|following(-sibling))
JavaScript Injection	<\s*script\b[^>]*>[^<]+<\s*/\s*script\s*>
Java Exception Injection	.*Exception in thread.*

NOTE: Because we configure policies in XML, your Regular Expressions must be URL Encoded.

```
[\s]*((delete)|(exec)|(drop\s*table)|(insert)|(shutdown)|(update)|(\bor\b))
```

becomes

```
%5B%5Cs%5D*((delete)%7C(exec)%7C(drop%5Cs*table)%7C(insert)%7C(shutdown)%7C(update)
%7C(%5Cbor%5Cb))
```

Sample Policy: Json Path

- Use the Expression tag for Json Path
- Use any valid Regex in the Pattern section

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RegularExpressionProtection async="false" continueOnError="false" enabled="true"
name="regular-expression-protection">
    <DisplayName>Regular Expression Protection</DisplayName>
    <FaultRules/>
    <Properties/>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <JSONPayload>
        <JSONPath>
            <Pattern ignoreCase="false"><\s*script\b[^>]*>[^<]+<\s*/\s*script\s*></Pattern>
            <Expression>$.</Expression>
        </JSONPath>
    </JSONPayload>
</RegularExpressionProtection>
```

Sample Policy: Query Parameters

- QueryParams are individually defined
- More than one Pattern can be used

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RegularExpressionProtection async="false" continueOnError="false" enabled="true"
name="regular-expression-protection">
    <DisplayName>Regular Expression Protection</DisplayName>
    <FaultRules/>
    <Properties/>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <QueryParam name=”testme">
        <Pattern>some pattern</Pattern>
        <Pattern>some other pattern</Pattern>
    </QueryParam>
    </JSONPayload>
</RegularExpressionProtection>
```

JSON Payload Protection Details

Minimizes the risk posed by content-level attacks by enabling you to specify limits on various JSON structures, such as arrays and strings.

Configuration Details

- ArrayElementCount

Specifies the Maximum number of Elements allowed in an Array

- ContainerDepth

Specifies the Maximum depth

- ObjecyEntryCount

Specifies the Maximum number of entries

- ObjectEntryNameLength

Specifies the Maximum string length of a name

- StringValueLength

Specifies the Maximum length of a String

- Source

Specifies the source of the data you want validated (typically request)

JSON Payload Protection Details

- JSON attacks attempt to use structures that overwhelm JSON parsers to try to crash a service
- The **JSONThreatProtection** policy minimizes the risk by specifying limits on JSON structure and string lengths
- <ContainerDepth> – maximum nesting depth

```
{
  "depth1": "value"
}
```

```
{
"depth1": { "depth2": "value" }
}
```

- <ArrayElementCount> – maximum number of elements in an array
- <ObjectEntryCount> – maximum number of entries in an object

```
{
  "entry1": "value",
  "entry2": { … },
  "entry3": [ … ]
}
```

- <ObjectEntryNameLength> – maximum string length for an object entry

```
{
  "thisIsAReallyLongEntryNameAndProbablyTooLong": "value"
}
```

Implementing Threat Protection – JSON

- <StringValueLength> – maximum string length for an entry's value

```
{
  "entry": "Value string lengths are often long, which is generally OK."
}
```

- The JSONThreatProtection policy will raise a fault if any of the supplied constraints are violated.
  - Choose these constraints carefully – don't block legitimate requests
  - Threat protection settings are often different for different APIs

Code: JSON-Threat-Protection

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<JSONThreatProtection async="false" continueOnError="false" enabled="true" name="JSON-Threat-Protection">
    <DisplayName>JSON Threat Protection</DisplayName>
    <FaultRules/>
    <Properties/>
    <ArrayElementCount>20</ArrayElementCount>
    <ContainerDepth>10</ContainerDepth>
    <ObjectEntryCount>15</ObjectEntryCount>
    <ObjectEntryNameLength>50</ObjectEntryNameLength>
    <Source>request</Source>
    <StringValueLength>500</StringValueLength>
</JSONThreatProtection>

```

XMLPayload Protection Details

Address XML vulnerabilities and minimize attacks on your API. Optionally, detect XML payload attacks based on configured limits. Screen against XML threats using the following approaches:

Validate messages against an XML schema (.xsd)
Evaluate message content for specific blacklisted keywords or patterns
Detect corrupt or malformed messages before those messages are parsed

Configuration Details

- Limits
  - NameLimits
    - Element
    - Attribute
    - Namespace
    - ProcessiingInstructionTarget (<**?xml**)
  - StructuralLimits
    - Depth
    - AtttributteCountPerElement
    - NamespaceCountPerElement
    - ChildCount
  - Value Limits
- Source

Specifies the source of the data you want validated (typically request)

Implementing Threat Protection - XML

- XML attacks attempt to send invalid XML documents that can compromise back end servers or send xml so complex that it causes the server to use more resources in an attempt to crash the server
- Like the JSONThreatProtection policy, the **XMLThreatProtection** policy is used to check the XML for certain **Structural limits**, **value limits** and **name limits**

Code: XML-Threat-Protection

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XMLThreatProtection async="false" continueOnError="false" enabled="true"  name="XML-Threat-Protection">
    <DisplayName>XML Threat Protection</DisplayName>
    <FaultRules/>
    <Properties/>
    <NameLimits>
        <Element>10</Element>
        <Attribute>10</Attribute>
        <NamespacePrefix>10</NamespacePrefix>
        <ProcessingInstructionTarget>10</ProcessingInstructionTarget>
    </NameLimits>
    <Source>request</Source>
    <StructureLimits>
        <NodeDepth>5</NodeDepth>
        <AttributeCountPerElement>2</AttributeCountPerElement>
        <NamespaceCountPerElement>3</NamespaceCountPerElement>
        <ChildCount includeComment="true" includeElement="true" includeProcessingInstruction="true" includeText="true">3</
    </StructureLimits>
    <ValueLimits>
        <Text>15</Text>
        <Attribute>10</Attribute>
        <NamespaceURI>10</NamespaceURI>
        <Comment>10</Comment>
        <ProcessingInstructionData>10</ProcessingInstructionData>
    </ValueLimits>
</XMLThreatProtection>

```

Implementing Threat Protection – Message Validation

- The **MessageValidation** policy can be used to reject invalid messages
  - SOAP message version and root element can be validated

Code: Message-Validation-Policy

```
<MessageValidation enabled="true" continueOnError="false" async="false" name="Message-Validation-Policy">
    <SOAPMessage version="1.2"/>
    <Element namespace="urn:ebay:apis:eBLBaseComponents">GetAccountRequest</Element>
    <Source>request</Source>
</MessageValidation>
```

  - SOAP and XML messages can use WSDL or schema (XSD) validation

Code: Message-Validation-Policy

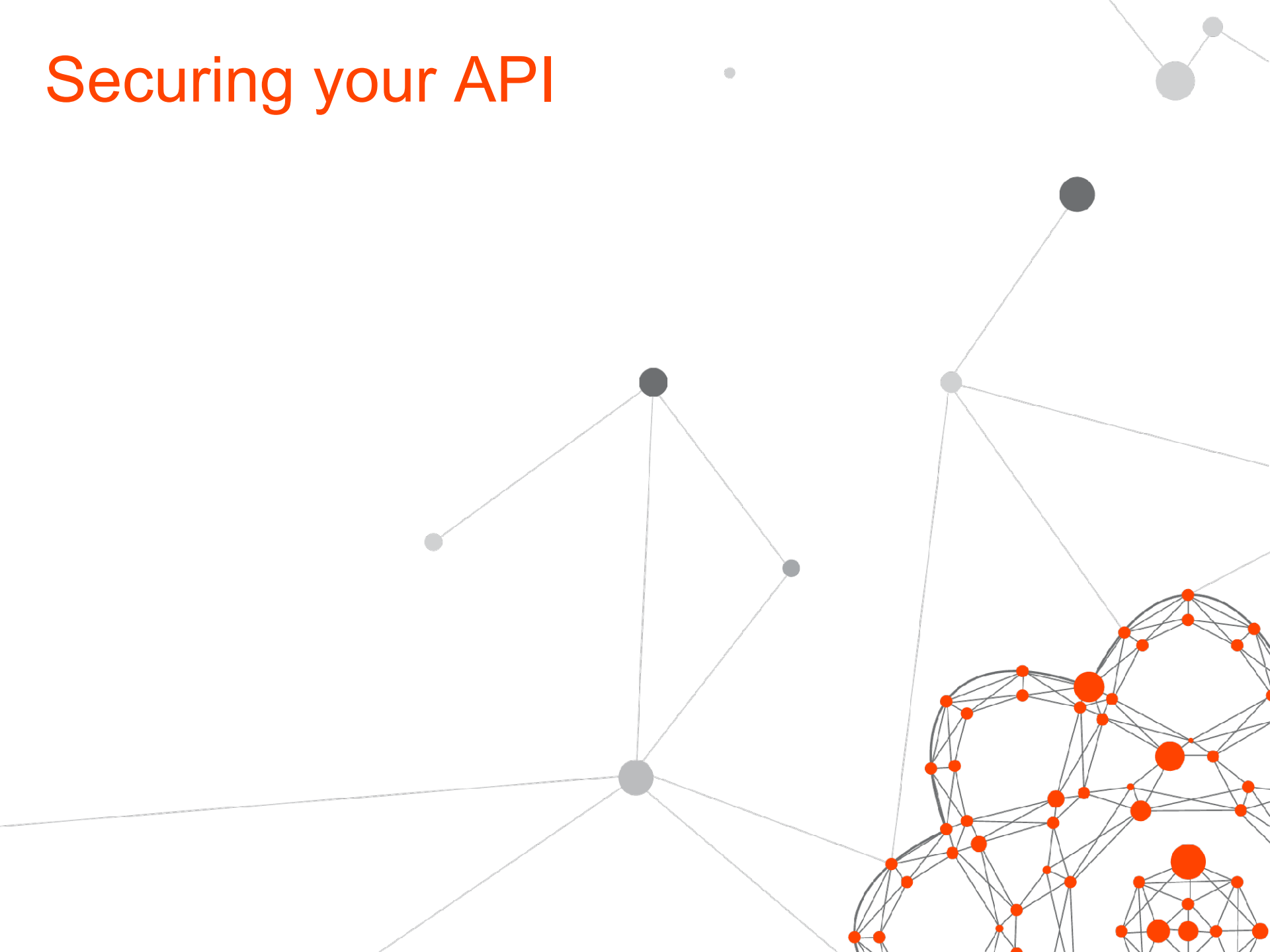
```
<MessageValidation enabled="true" continueOnError="false" async="false" name="Message-Validation-Policy">
    <ResourceURL>xsd://GetUserRecord.xsd</ResourceURL>
    <Source>request</Source>
</MessageValidation>
```

  - If no <ResourceURL>, XML and JSON messages will still be validated for well-formedness

Code: Message-Validation-Policy

```
<MessageValidation enabled="true" continueOnError="false" async="false" name="Message-Validation-Policy">
    <Source>request</Source>
</MessageValidation>
```

Securing your API



Authentication and Authorization vs. Identity Tracking

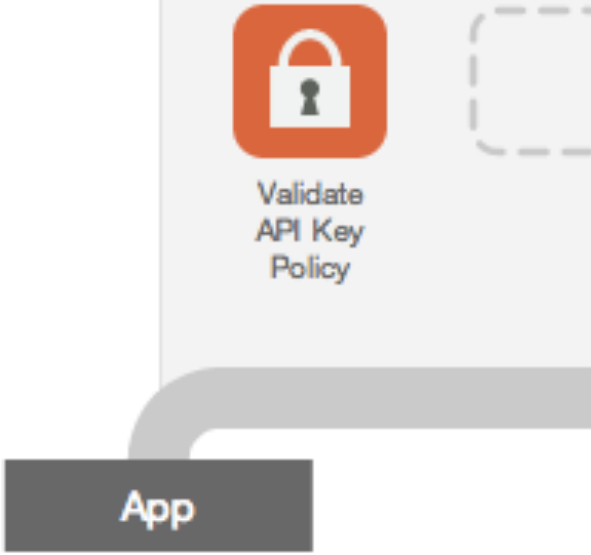
- **Authentication** involves validating application and/or user credentials ("who you are")

- **Authorization** involves determining what the application/user can do ("what you can do")

- **Identity tracking** is concerned with tracking usage by an application and/or user
  - Not concerned with authorizing access, just who/which app is using service
  - Applications generally tracked using API keys
  - Users tracked via credentials
  - Other identity tracking can be done via IP address and Host header

Verify API Key

Application key verification blocks unwelcome traffic and populates variables for other policies (such as Quota) and sets variables for tracking in Analytics

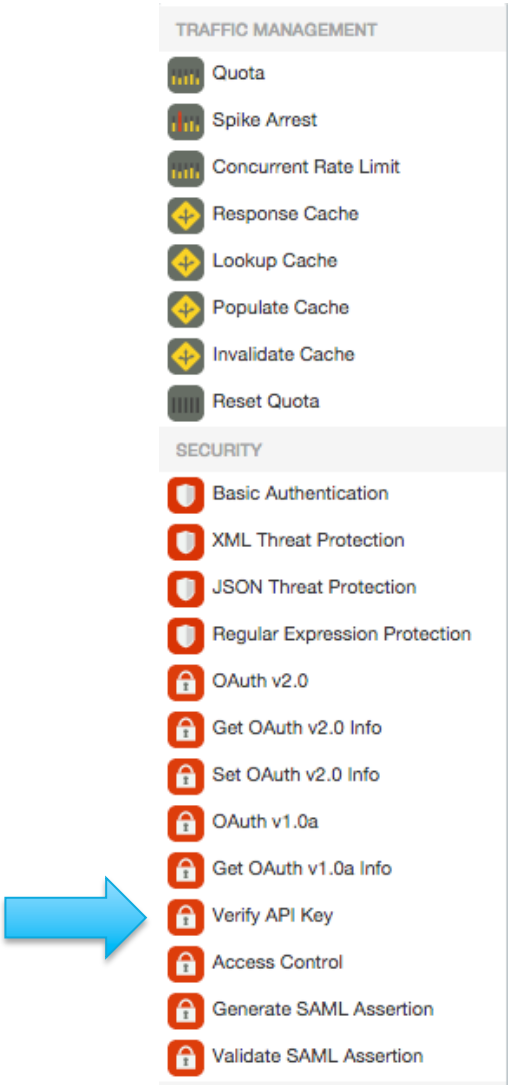



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VerifyAPIKey enabled="true" continueOnError="false" async="false"
name="VerifyAPIKey-1">
    <DisplayName>Verify API Key 1</DisplayName>
    <FaultRules/>
    <Properties/>
    <APIKey ref="request.queryparam.apikey"></APIKey>
</VerifyAPIKey>
```

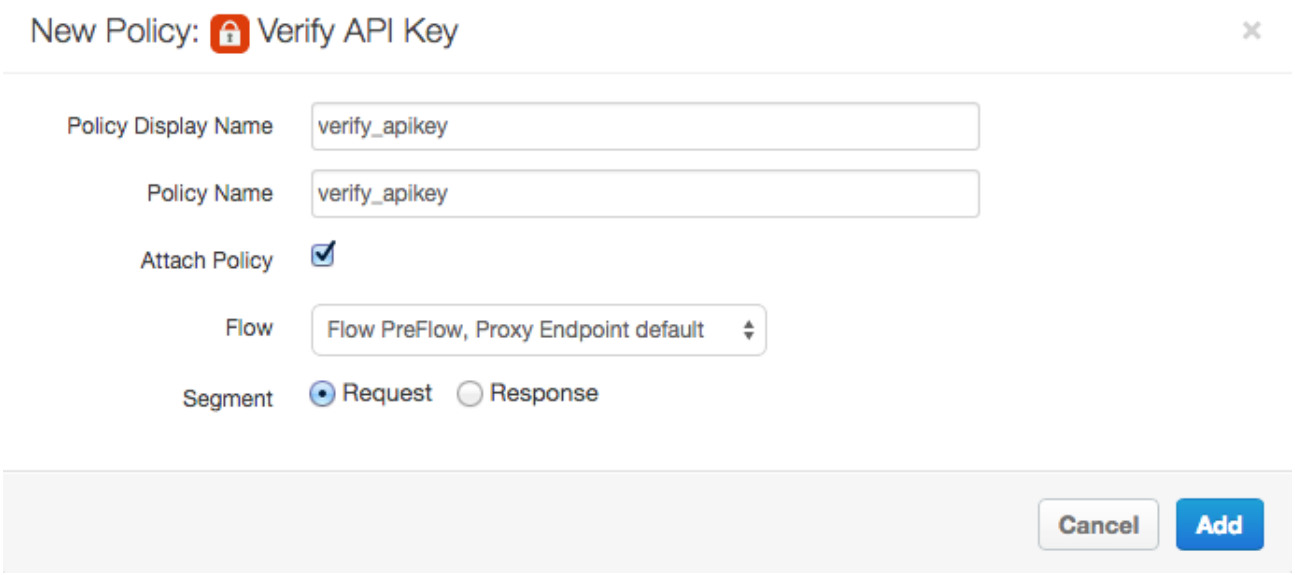
Configurable query param (default “apikey”)

Add Verify API Key

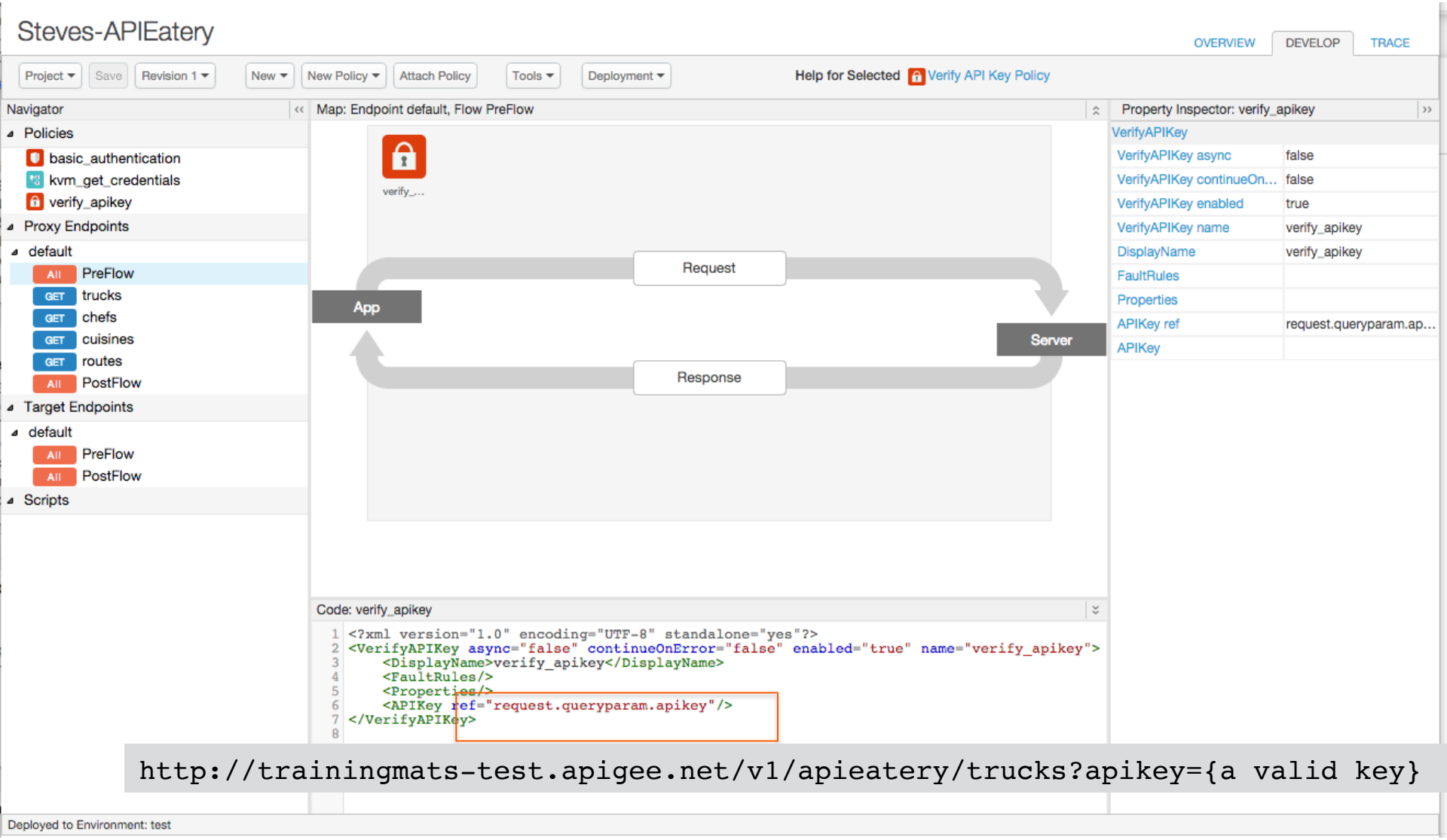
Select Verify API Key



Name and Select the Proxy Endpoint (Preflow) from the flows, then press Add



Apikey Validation




```
http://trainingmats-test.apigee.net/v1/apieatery/trucks?apikey={a valid key}
```

Apikey in App

StevesApp

Regenerate Key | Edit | Delete

Developer App Details

Display Name	StevesApp
Registered	Sep 10, 2014 1:59:55 PM
Developer	Steven Richardson (srichardson@apigee.com)
Callback URL	
Notes	

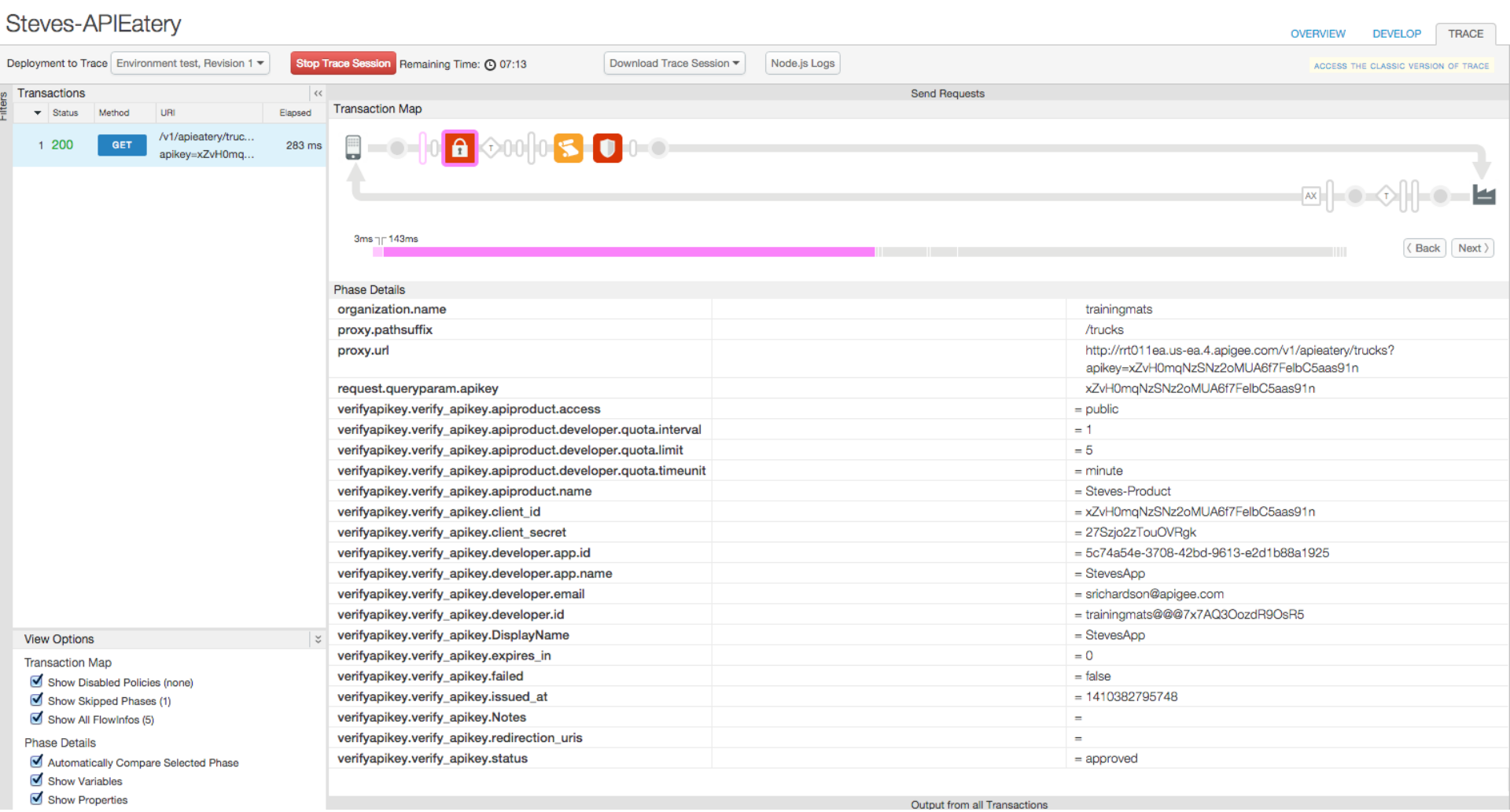
Products

Product	Status	Consumer Key	Consumer Secret
Steves Product	Approved	xZvH0mqNzSNz2oMUA6f7FelbC5aas91n Hide	27Szjo2zTouOVRgk Hide

Custom Attributes

Name ▲	Value

Apikey Attributes



Day2 – Lab 8 Apikey

- Create a new Product
- Create a new APP
  - Add a developer to the APP
  - Add the product above to APP
  - Go to the APP and grab the apikey
- Create “Verify API Key” policy in Preflow
- Test the API with postman
  - Notice the request will fail as no apikey is provided
- Provide apikey in the request query param
  - ..?apikey=blah
- Test the API again with postman

APIKey vs. Access Token

Common Authentication Access Patterns

APIKey

```
http://api.yourcompany.com/target?
apikey=45c78ece5b77647854a84dfb4ba96dc8
```

Access Token

```
/target?access_token=4WCAchNNtVyK8JsACl1HP7ml
```

Bearer Token

```
Authorization: Bearer 4WCAchNNtVyK8JsACl1HP7ml
```

Using Client Credentials Grant

Generate Token

Developer passes Base64 encoded Basic Auth request (client_id:client_secret) to generate access_token

Request:

```
http://api.yourcompany.com/oauth/token?
grant_type=client_credentials
```

Header:

```
Authorization: Basic
d293IHlvdSByZWFsbHkgdG9vayB0aGUgdGltZSB0byBkZWNvZGU
gdGhpcz8g
```

Response:


```
{
    "issued_at": "1393889380896",
    "application_name": "030fdcea-cf97-12084aea513c",
    "scope": "",
    "status": "approved",
    "api_product_list": "[weathe
    "expires_in": ”86400",
    "developer.email": "tesla@weathersample.com",
    "organization_id": "0",
    "token_type": "BearerToken",
    "client_id": "RqBca4HGxdyaDM6AAPIHfQ53kLLIGFMf",
    "access_token": "4WCAchNNtVyK8JsACl1HP7mlWW1X",
    "organization_name": "jokeindex",
    "refresh_token_expires_in": "0",
    "refresh_count": "0"
}
```


Expires in Seconds

The Generate Token Policy

- Separate flow for security
- Use Callouts to populate custom attributes

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="oauth-generate-token">
    <DisplayName>OAuth Generate  Token</DisplayName>
    <Operation>GenerateAccessToken</Operation>
    <ExpiresIn>86400000</ExpiresIn>
    <SupportedGrantTypes>
      <GrantType>client_credentials</GrantType>
    </SupportedGrantTypes>
    <GrantType>request.queryparam.grant_type</GrantType>
  <GenerateResponse/>
</OAuthV2>
```

Expires in Milliseconds

OAuth V 2.0

OAuth Basic Concepts

- **OAuth 2.0** is a protocol that allows clients to grant access to server resources to another entity without sharing credentials
- **Client IDs and Secrets** are used to identify and authenticate applications (application's consumer key and consumer secret)
- **Tokens** are issued to allow access to specific resources for a specified period of time and may be revoked by the user that granted permission or by the server that issued the token
- **Scopes** can be used to limit the access for a given token, granting permission only for the operations that are necessary
- Four different **Grant Types** specify the different authentication usage scenarios OAuth supports
- Tokens must be protected, and OAuth 2.0 requires that all API traffic be sent via **SSL**

Access Tokens

Access Tokens allow access to a protected resource for a specific application to perform only certain actions for a limited period of time.

Identification info from the requesting application **(client ID and secret)**

+

Resource owner credentials (if needed)

+

Optional information about what the application wants to do with the resource **(scope)**

=

Access Token and (optional) refresh token

In Apigee, access tokens are opaque strings with no encoded meaning. Access tokens are passed as Bearer tokens in an Authorization header.

Refresh Tokens

Refresh Tokens, if provided, represent a limited right to reauthorize the granted access by obtaining new access tokens.

Identification info from the requesting application **(client ID and secret)**

+

Refresh token

+

Optional information about what the application wants to do with the resource **(scope)**

=

Access Token

In Apigee, refresh tokens are opaque strings with no encoded meaning.

Scopes

Scopes identify what an application can do with the resources it is requesting access to. Scope names are defined by the authorization server and are associated with information that enables decisions on whether a given API request is allowed or not.

Apigee associates scope names to be matched with a combination of API resource path and verb. So, for example:

Scope 1: "READ"

- GET /photos
- GET /photos/{id}

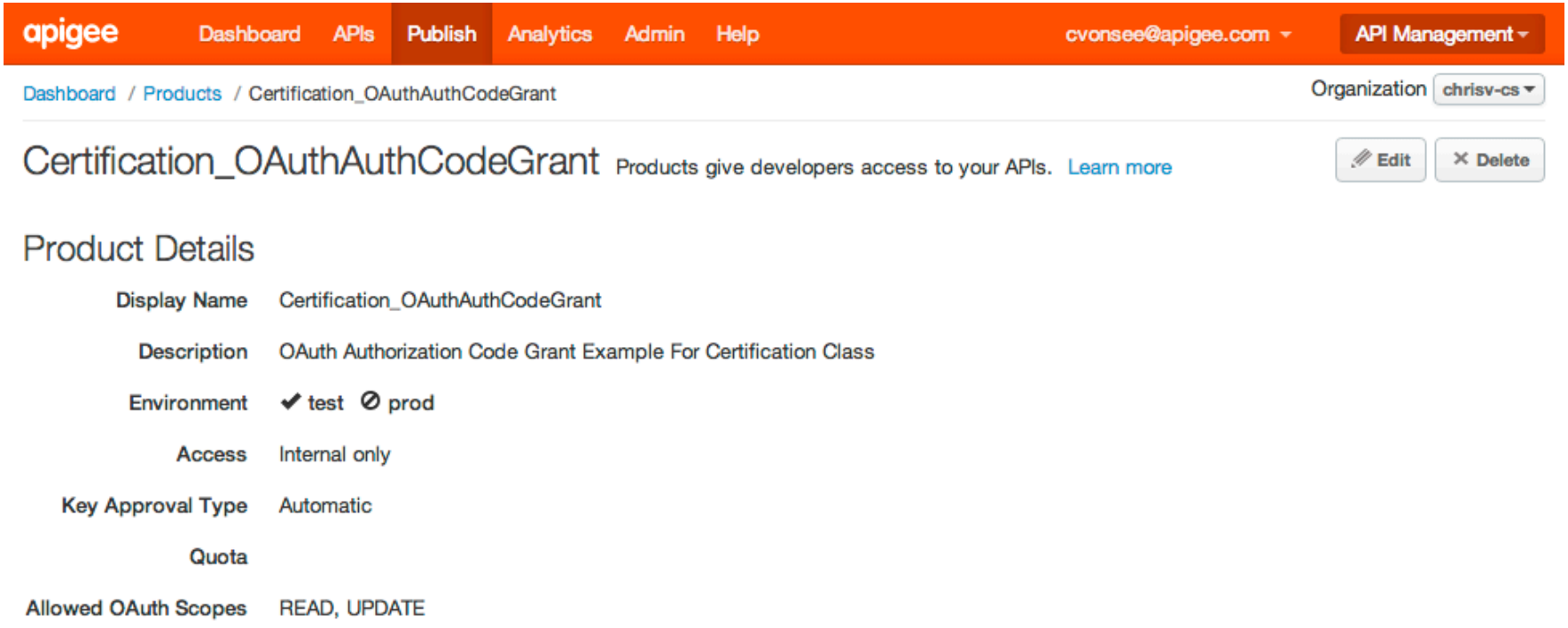
Scope 2: "UPDATE"

- GET /photos
- GET /photos/{id}
- POST /photos
- PUT /photos/{id}

When an application requests an access token, the scope names are optional.

Scopes

- The OAuth spec allows for an app to specify no scope on a token request, in which case you should either:
  - assign a default scope or no scopes (the usual case), or
  - reject the request
- One or more scopes can be specified in the API Product definition

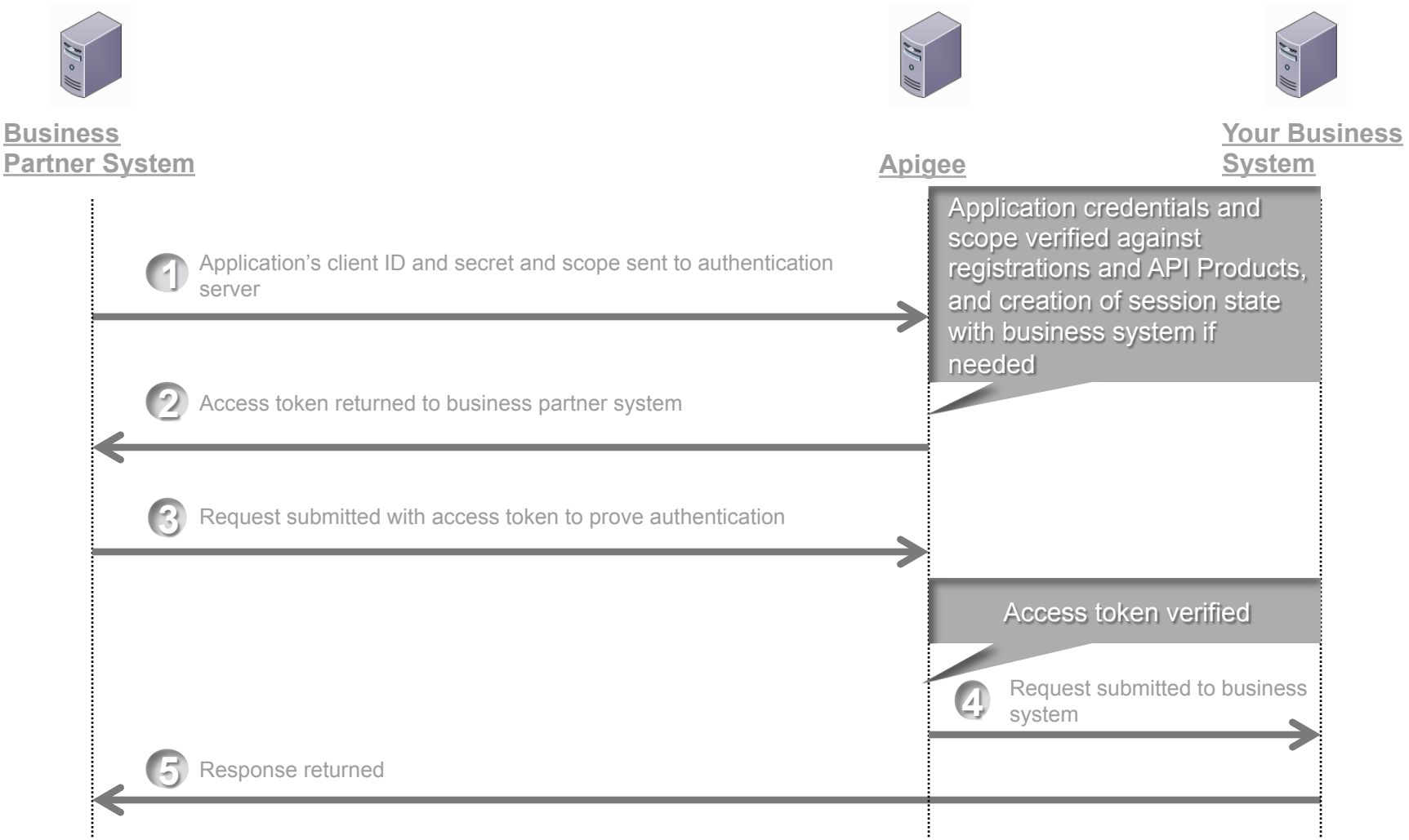


OAuth Grants Types

An **OAuth Grant** is a credential representing the resource owner's authorization. More often than not, we tend to think of grants in terms of the process used to obtain an access token.

Grant Type	Typical Use Case	Complex?
No specific resource owner is involved		
Client Credentials	Business system interactions, where resources being operated on are owned by the partner, not a particular user	No
A specific resource owner is involved		
Resource Owner Password Credentials	Resources are owned by a particular user and the requesting application is **trusted**	A bit
Authorization Code	Resources are owned by a particular user and the requesting application is **untrusted**	Very
Implicit	Resources are owned by a particular user, and the requesting application is an **untrusted browser-based app written in a scripting language such as JavaScript**	Very, and potentially insecure as well

Server-to-Server Communication using Client Credential Grant Type



Implementing OAuth: Client Credentials Grant

- Generate tokens for the client credentials grant using the OAuthV2 policy. Be sure to set the token endpoint to use SSL.

Code: GenerateAccessToken

```
<OAuthV2 name="GenerateAccessToken">
  <!-- This policy generates an OAuth 2.0 access token using the client_credentials grant type -->
  <Operation>GenerateAccessToken</Operation>
  <!-- This is in millseconds, so expire in an hour -->
  <ExpiresIn>3600000</ExpiresIn>
  <SupportedGrantTypes>
    <!-- This part is very important: most real OAuth 2.0 apps will want to use other
         grant types. In this case it is important to NOT include the "client_credentials"
         type because it allows a client to get access to a token with no user authentication -->
    <GrantType>client_credentials</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <GenerateResponse enabled="true"/>
</OAuthV2>
```

Implementing OAuth: Client Credentials Grant

- The OAuthV2 policy's "VerifyAccessToken" operation will validate the access token for subsequent requests **for all grant types**.

Code: VerifyOAuthToken

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="VerifyOAuthToken">
    <DisplayName>VerifyOAuthToken</DisplayName>
    <FaultRules/>
    <Operation>VerifyAccessToken</Operation>
</OAuthV2>
```

- Set the access token as the Bearer token in the Authorization header of the http request.

```
curl -H "Authorization: Bearer {access_token}"
http://myorg-test.apigee.net/v1/cc/oauth_cc_weather/forecastrss?w=12797282
```

Scopes

- You can generate tokens that are value only for a specific scope:

Code: OAuth-v20-1

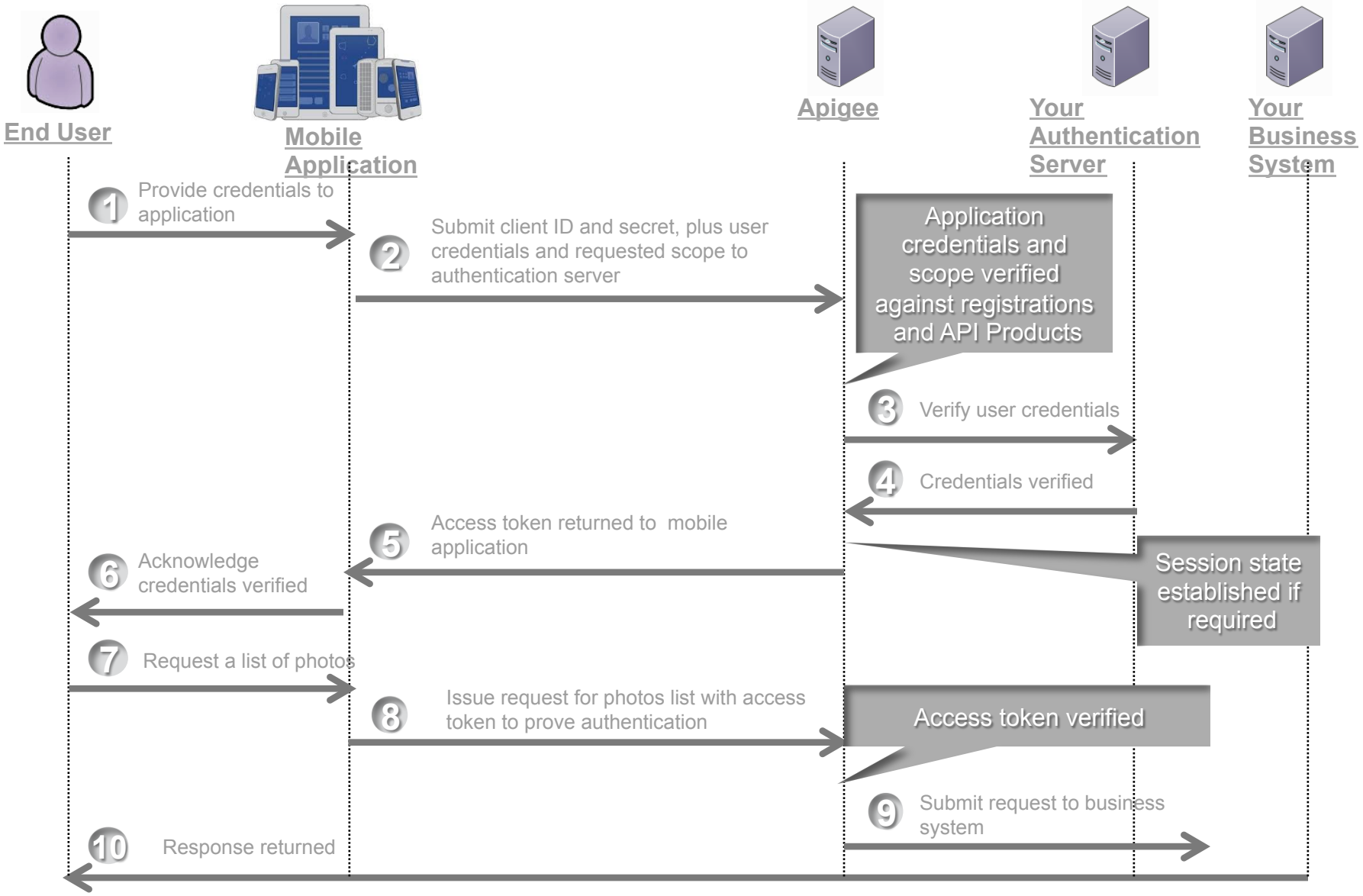
```
<OAuthV2 name="GenerateAccessToken">
  <Operation>GenerateAccessToken</Operation>
  <ExpiresIn>1000</ExpiresIn>
  <GenerateResponse />
  <SupportedGrantTypes>
   <GrantType>authorization_code</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <Code>request.queryparam.code</Code>
  <ClientId>request.queryparam.client_id</ClientId>
  <RedirectUri>request.queryparam.redirect_uri</RedirectUri>
  <Scope>request.queryparam.scope</Scope>
</OAuthV2>
```

- and verify that tokens are valid for a specific scope:

Code: OAuth-v20-1

```
<OAuthV2 name="ValidateOauthScopePolicy">
  <Operation>VerifyAccessToken</Operation>
  <Scope>READ WRITE</Scope>
</OAuthV2>
```

Accessing End User Photos using a Trusted Application with Password Grant



Implementing OAuth: Resource Owner Password Credentials Grant

- The resource owner password grant is similar to the client credentials grant type, but with an extra step to validate the user credentials.
- Create this endpoint and use the OAuthV2 policy to generate an access token.

Code: GenerateAccessToken

```
<OAuthV2 name="GenerateAccessToken">
  <!-- This policy generates an OAuth 2.0 access token using the client_credentials grant type -->
  <Operation>GenerateAccessToken</Operation>
  <!-- This is in millseconds, so expire in an hour -->
  <ExpiresIn>3600000</ExpiresIn>
  <SupportedGrantTypes>
    <!-- This part is very important: most real OAuth 2.0 apps will want to use other
         grant types. In this case it is important to NOT include the "client_credentials"
         type because it allows a client to get access to a token with no user authentication -->
    <GrantType>password</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.formparam.grant_type</GrantType>
  <GenerateResponse enabled="true"/>
  <UserName>request.formparam.username</UserName>
  <PassWord>request.formparam.password</PassWord>
</OAuthV2>
```

Responses: Client Credentials vs Password Grant

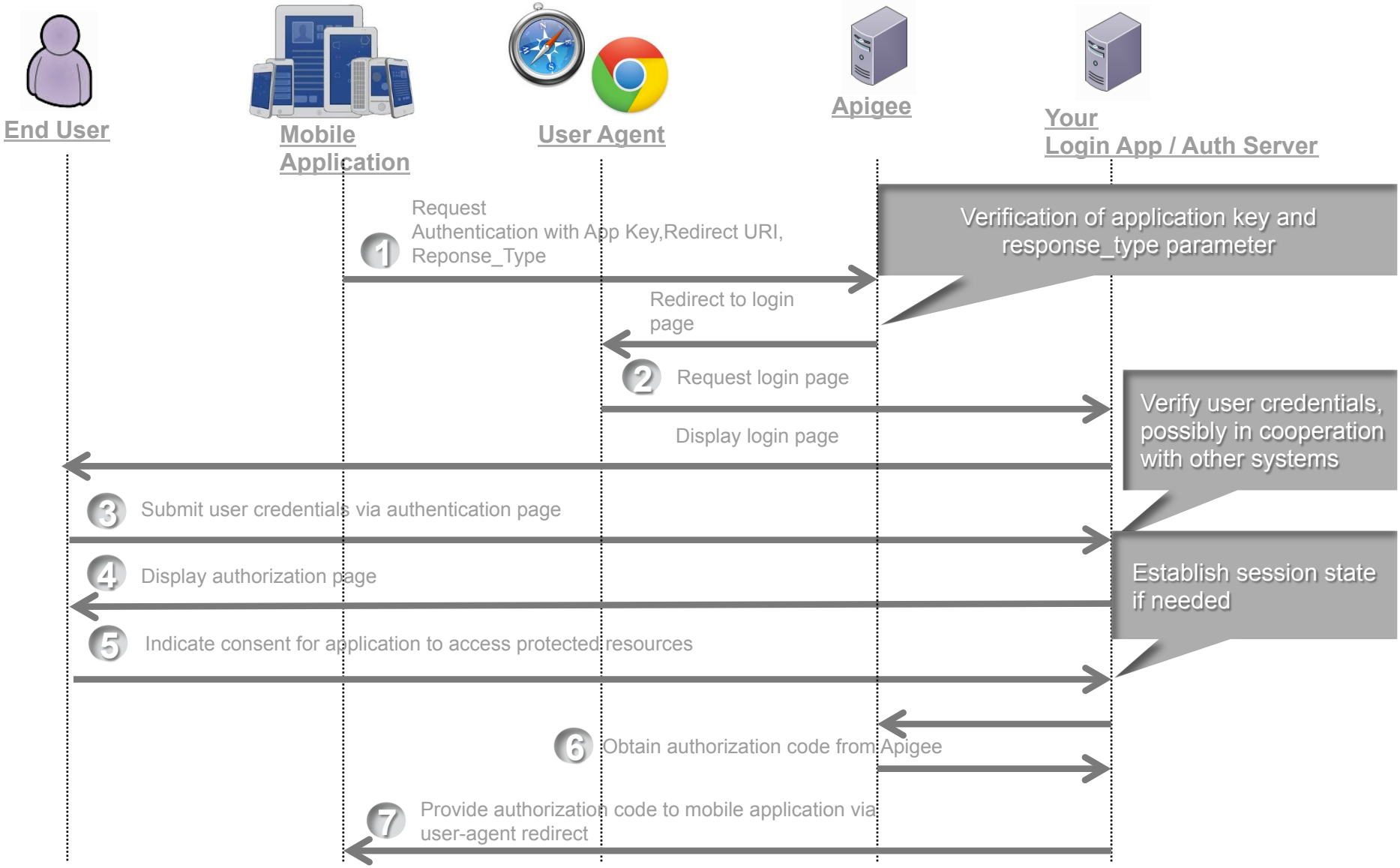
Client Credentials Grant


```
{
  "issued_at" : "1407513671919",
  "application_name" : "26c855a9-c485-4318-accc-7e3f533a154c",
  "scope" : "",
  "status" : "approved",
  "api_product_list" : "[Certification_OAuthClientCredentialsWeather]",
  "expires_in" : "3599",
  "developer.email" : "certifieddev@apigee.com",
  "organization_id" : "0",
  "token_type" : "BearerToken",
  "client_id" : "vn0zG4cnSWaWIzdwBZgnREI1NGORDXXz",
  "access_token" : "2CsgxkPqfNtCSAZ5qGEI9x5dGdvV",
  "organization_name" : "chrisv-cs",
  "refresh_token_expires_in" : "0",
  "refresh_count" : "0"
}
```

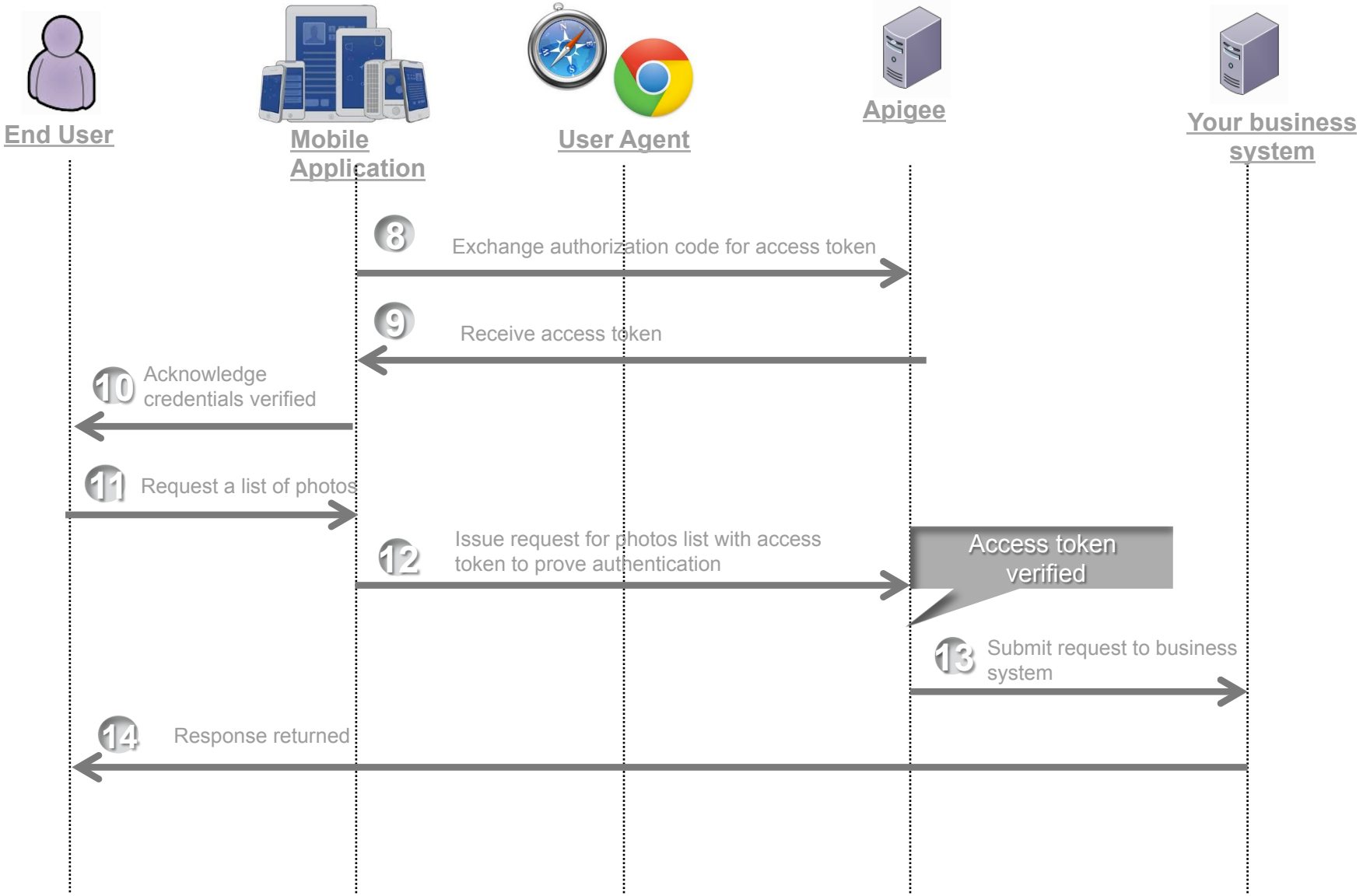

Password Grant


```
{
  "issued_at" : "1407513709051",
  "scope" : "",
  "application_name" : "26c855a9-c485-4318-accc-7e3f533a154c",
  "refresh_token_issued_at" : "1407513709051",
  "status" : "approved",
  "refresh_token_status" : "approved",
  "api_product_list" : "[Certification_OAuthClientCredentialsWeather]",
  "expires_in" : "3599",
  "developer.email" : "certifieddev@apigee.com",
  "organization_id" : "0",
  "token_type" : "BearerToken",
  "refresh_token" : "HsnXmyIQqmJJQrFVdevmVztGGASUfBfz",
  "client_id" : "vn0zG4cnSWaWIzdwBZgnREI1NGORDXXz",
  "access_token" : "GRQAJcgSFZcklbIUxfoUaYFW2ROd",
  "organization_name" : "chrisv-cs",
  "refresh_token_expires_in" : "0",
  "refresh_count" : "0"
}
```

Accessing End User Photos using an Untrusted Application with an Authorization Code Grant



Accessing End User Photos using an Untrusted Application with an Authorization Code Grant



Implementing OAuth: Authorization Code Grant

- Generate authorization code

Code: GenerateAuthorizationCode

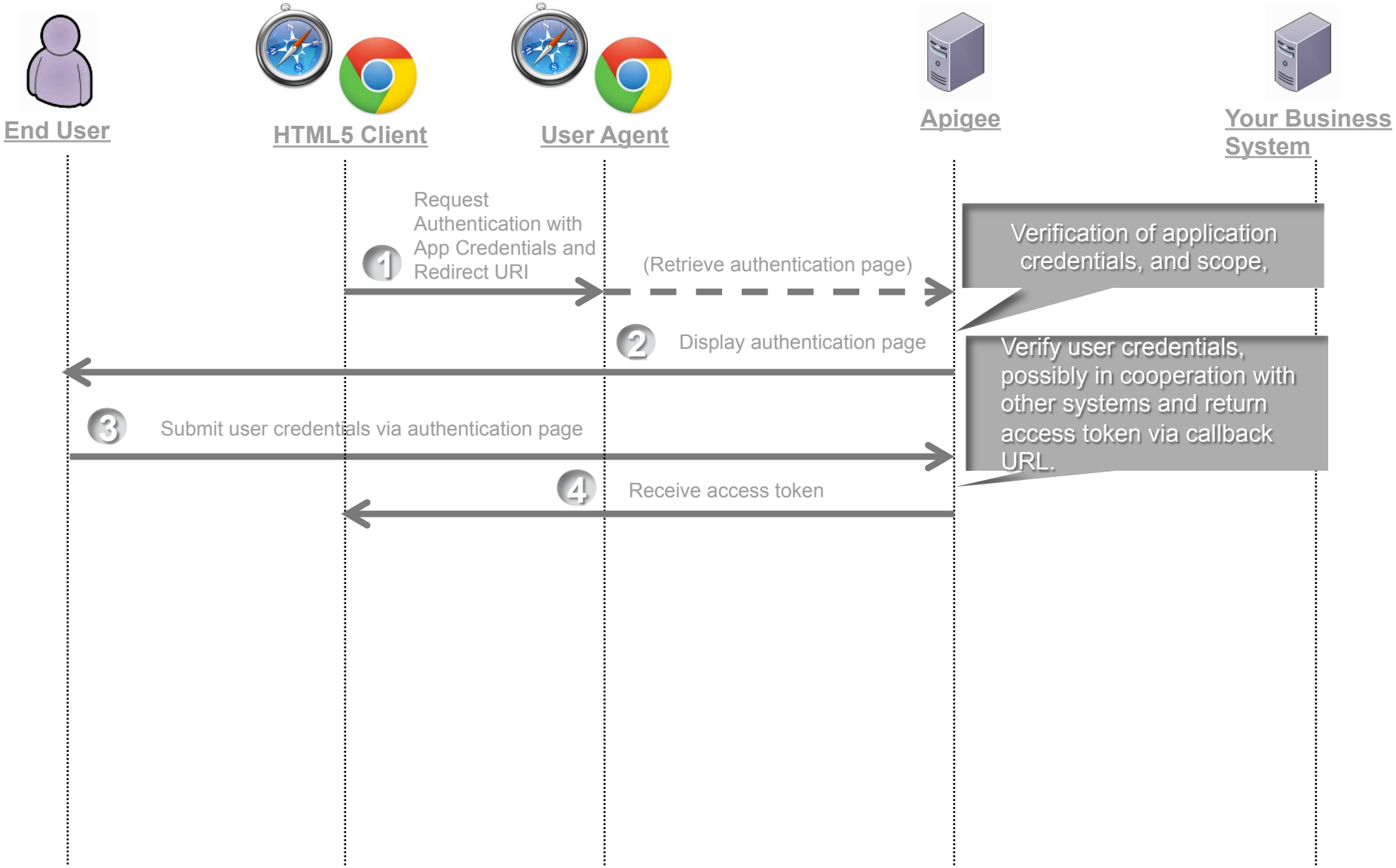
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 enabled="true" continueOnError="false" async="false" name="GenerateAuthorizationCode">
    <FaultRules/>
    <Properties/>
    <Operation>GenerateAuthorizationCode</Operation>
    <GenerateResponse/>
</OAuthV2>
```

- Exchange authorization code for an access token

Code: GenerateAccessToken

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 name="GenerateAccessToken">
    <FaultRules/>
    <Properties/>
    <ExpiresIn>3600000</ExpiresIn>
    <SupportedGrantTypes>
      <GrantType>authorization_code</GrantType>
    </SupportedGrantTypes>
    <GenerateResponse/>
</OAuthV2>
```

Untrusted Application Obtaining an Access Token with Implicit Grant



Refreshing tokens

- Refresh tokens can be used to obtain new access tokens for the original end user:

Code: RefreshToken

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="RefreshToken">
    <DisplayName>RefreshToken</DisplayName>
    <FaultRules/>
    <Properties/>
    <Attributes/>
    <Operation>RefreshAccessToken</Operation>
  <ReuseRefreshToken>true</ReuseRefreshToken>
  <ExpiresIn>3600000</ExpiresIn>
  <GrantType>request.formparam.grant_type</GrantType>
  <RefreshToken>request.formparam.refresh_token</RefreshToken>
    <GenerateResponse enabled="true"/>
    <Tokens/>
</OAuthV2>
```

```
 curl -v -u "vn0zG4cnSWaWIzdwBZgnREI1NGORDXXz:uZt7LeDbq7vX90NP"
-d "grant_type=refresh_token&refresh_token=LRYvVXXgWwbuEZivoe8XvZE2WPdzX9fp"
"http://chrisv-cs-test.apigee.net/v1/cc/oauth_rtg/refresh_accesstoken"
```

Invalidating access and refresh tokens

- Access and refresh tokens can be invalidated via policies:

Code: InvalidateToken

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="InvalidateToken">
    <DisplayName>InvalidateToken</DisplayName>
  <Operation>InvalidateToken</Operation>
  <Tokens>
    <Token type="accesstoken" cascade="true">request.queryparam.access_token</Token>
    <Token type="refreshtoken" cascade="true">request.queryparam.refresh_token</Token>
  </Tokens>
</OAuthV2>
```

- “cascade=true” propagates invalidation to all related access and refresh tokens

Access token attributes

- Access/refresh tokens and authorization codes can contain attributes

Code: generateapigeeaccesstoken_ccg

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="generateapigeeaccesstoken_ccg">
    <DisplayName>GenerateApigeeAccessToken_CCG</DisplayName>
    <FaultRules/>
    <Properties/>
    <Attributes/>
  <Operation>GenerateAccessToken</Operation>
  <SupportedGrantTypes>
    <GrantType>client_credentials</GrantType>
  </SupportedGrantTypes>
  <GrantType>credentials.grant_type</GrantType>
  <ExpiresIn ref="expiresIn_string">3600</ExpiresIn>
  <Attributes>
    <Attribute name="appServicesOrgName" ref="verifyapikey.verifyapikeyfromauthheader.DATA_STORE_ORG_NAME" display="false"></Attribute>
    <Attribute name="appServicesAppName" ref="verifyapikey.verifyapikeyfromauthheader.DATA_STORE_APP_NAME" display="false"></Attribute>
    <Attribute name="appServicesConsumerKey" ref="verifyapikey.verifyapikeyfromauthheader.DATA_STORE_APP_CONSUMER_KEY" display="false"></Attribute>
    <Attribute name="appServicesAccessToken" ref="appServicesTokenResponse.accessToken" display="false"></Attribute>
    <Attribute name="appServicesExpiresIn" ref="expiresIn_string" display="false"></Attribute>
  </Attributes>
</OAuthV2>
```

Using GetOAuthV2Info to retrieve access token attributes

- GetOAuthV2Info used to retrieve general info about an access token
- GetOAuthV2Info is usually not needed – the OAuthV2 VerifyAccessToken operation now populates similar variables
- http://snap.apigee.com/ApigeeOAuthVariables

Code: GetAccessTokenInfo

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <GetOAuthV2Info async="false" continueOnError="false" enabled="true" name="GetAccessTokenInfo">
3     <DisplayName>GetAccessTokenInfo</DisplayName>
4     <AccessToken ref="request.queryparam.token"></AccessToken>
5 </GetOAuthV2Info>
```

Phase Details

Variables

apigee.access_token		= 15fVAgbJc2a5rzGeApdlZ8oemR5i
apigee.apiproduct.name		= Certification_OAuthClientCredentialsWeather
apigee.client_id		= vn0zG4cnSWaWlzdwBZgnREI1NGORDXXz
apigee.developer.app.name		= CertificationOAuthCCWeather
apigee.developer.email		= certifieddev@apigee.com
apigee.developer.id		= chrisv-cs@@@3MGS3gmo7zxrEdTr
apiproxy.name		Certification_OAuthGetSetInfo
environment.name		test
flow.resource.name		
oauthV2.failed		= false

Using SetOAuthV2Info to set access token attributes

- SetOAuthV2Info can be used to set attributes on an access token…

Code: SetAccessTokenInfo

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SetOAuthV2Info async="false" continueOnError="false" enabled="true" name="SetAccessTokenInfo">
    <DisplayName>SetAccessTokenInfo</DisplayName>
    <AccessToken ref="request.queryparam.token"></AccessToken>
    <Attributes>
      <Attribute name="myAttribute1">foo</Attribute>
      <Attribute name="myAttribute2">baz</Attribute>
  </Attributes>
</SetOAuthV2Info>
```

Using SetOAuthV2Info to set access token attributes

- … which can then be retrieved by validating the token

Variables

access_token		= 15fVAgbJc2a5rzGeApdlZ8oemR5i
accesstoken.myAttribute1		= foo
accesstoken.myAttribute2		= baz
apigee.access_token		= 15fVAgbJc2a5rzGeApdlZ8oemR5i
apigee.apiproduct.name		= Certification_OAuthClientCredentialsWeather
apigee.client_id		= vn0zG4cnSWaWlzdwBZgnREI1NGORDXXz
apigee.developer.app.name		= CertificationOAuthCCWeather
apigee.developer.email		= certifieddev@apigee.com
apigee.developer.id		= chrisv-cs@@@3MGS3gmo7zxrEdTr
apiproduct.access		= internal
apiproduct.name		= Certification_OAuthClientCredentialsWeather
apiproxy.name		Certification_OAuthGetSetInfo
app		= 26c855a9-c485-4318-accc-7e3f533a154c
client_id		= vn0zG4cnSWaWlzdwBZgnREI1NGORDXXz
developer.app.id		= 26c855a9-c485-4318-accc-7e3f533a154c
developer.app.name		= CertificationOAuthCCWeather

More Useful OAuth information

- OAuthV2 policy usage: http://snap.apigee.com/ApigeeOAuthV2Policy
- OAuth flow variables: http://snap.apigee.com/ApigeeOAuthVariables
- OAuth error code reference: http://snap.apigee.com/ApigeeOAuthErrorCodes

Using the Access Token

Authentication

APIKey

```
http://api.yourcompany.com/resource?
apikey=45c78ece5b77647854a84dfb4ba96dc8
```

Access token

```
/resource?
access_token=45c78ece5b77647854a84dfb4ba96dc8
```

Bearer Token

```
Authorization: Bearer
YWMtZoQdDo02EeOEVMtbc899TQAAAUQebNlF1vQ8yLtX5I
SQiTPvfWgLfUSsFVE
```

Using Access_tokens

Pass the access_token as a Bearer Token

Request:
```
http://trainingmats-test.apigee.net/v1/apieatery/trucks
```
Header:
```
Authorization: Bearer 4WCAchNNtVyK8JsACl1HP7mlWW1X
```

The Verify Access Token Policy

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OAuthV2 async="false" continueOnError="false" enabled="true" name="validate-oauth">
    <DisplayName>Validate OAuth</DisplayName>
    <FaultRules/>
    <Properties/>
    <Attributes/>
    <ExternalAuthorization>false</ExternalAuthorization>
    <Operation>VerifyAccessToken</Operation>
    <SupportedGrantTypes/>
    <Tokens/>
</OAuthV2>
```

Day2 – Lab 9 Client Credential Grant

- Create a new resource /token
  - Name=token, method=GET, path=/token
- Create a “noTarget” Route rule in Proxy Endpoint

```
<RouteRule name="NoTarget">
    <Condition>(proxy.pathsuffix MatchesPath &quot;/token&quot;) and (request.verb = &quot;GET&quot;)</Condition>
</RouteRule>
```

- Create a new policy “Oauth v2.0” in /token resource

```
<ExternalAuthorization>false</ExternalAuthorization>
  <Operation>GenerateAccessToken</Operation>
  <SupportedGrantTypes>
      <GrantType>client_credentials</GrantType>
  </SupportedGrantTypes>
  <GrantType>request.queryparam.grant_type</GrantType>
  <GenerateResponse enabled="true"/>
  <Tokens/>
```

Day2 – Lab 9 Client Credential Grant Cont.

- Test using postman to make sure token is generated
  - http://orgname-envname.apigee.net/v0/apieatery/token?grant_type=client_credentials
  - Set header parameter, Authorization = basic B64(apikey:secret)

Day2 – Lab 9 Client Credential Grant Cont.

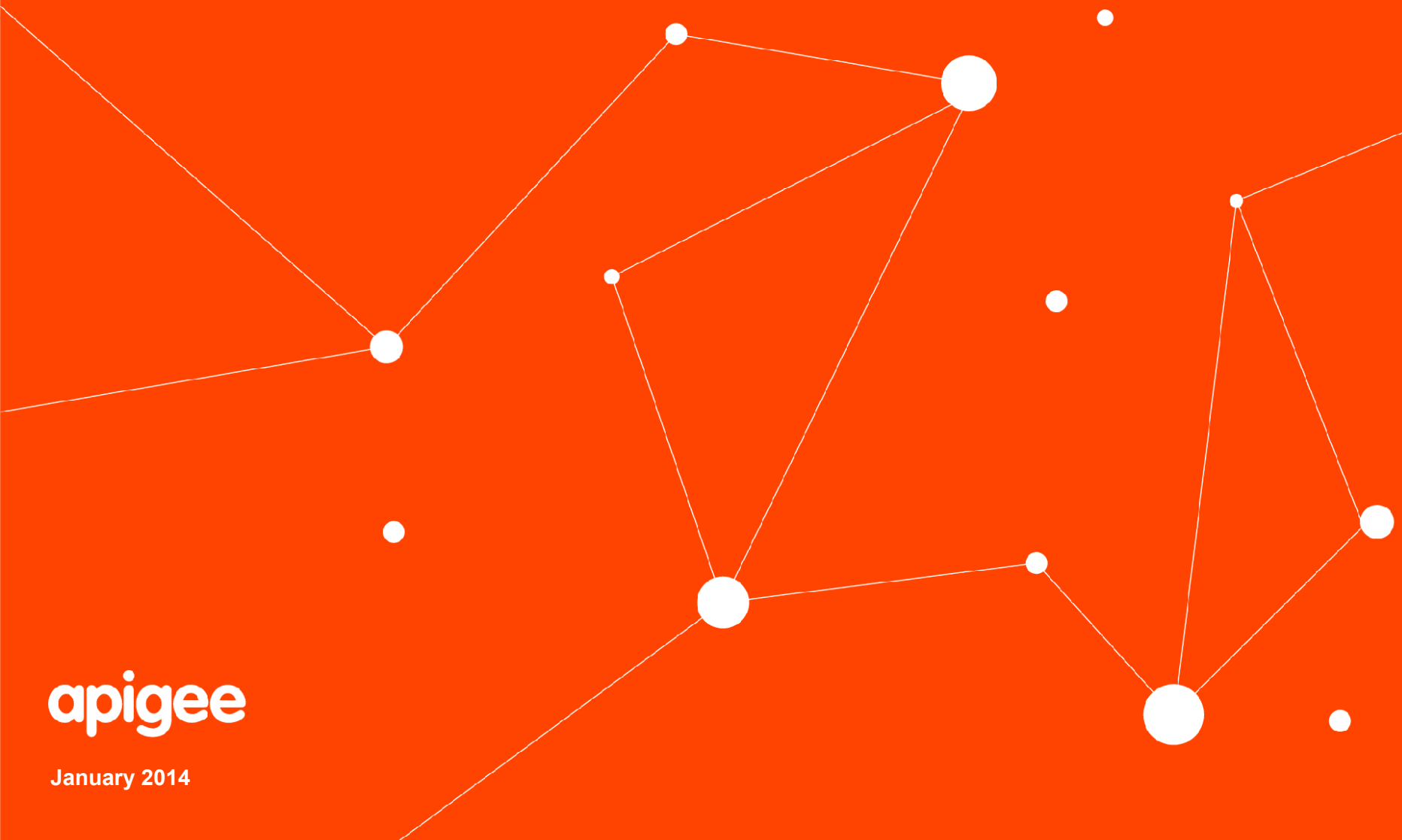
OPTIONAL LAB

- Create a OAuth2 policy in ratings

```
<OAuthV2 async="false" continueOnError="false" enabled="true" name="AuthenticateToken">
        <DisplayName>AuthenticateToken</DisplayName>
        <ExternalAuthorization>false</ExternalAuthorization>
        <Operation>VerifyAccessToken</Operation>
        <SupportedGrantTypes/>
        <GenerateResponse enabled="true"/>
        <Tokens/>
</OAuthV2>
```

- Use this token to send a request to */ratings
  - http://yourOrg-yourEnv.apigee.net/v0/apieatery/ratings
  - Set Authorization = Bearer token

Thank you



apigee

January 2014