JavaOne℠

Designing and Building Security into REST Applications

Paul C. Bryan
Sean Brydon
Aravindan Ranganathan

Sun Microsystems, Inc.

OpenSSO

Goals of our talk

Provide some practical tips and guidelines to enable you to to build REST services securely

Identify some different use cases and different requirements for security

Provide some anti-patterns and pitfalls — learn from our mistakes too!

Discuss some useful standards including a key emerging standard

Agenda

- **What is OpenSSO**
- Problem description
- REST overview
- REST security overview
- OpenSSO simple REST services APIs
- OAuth overview
- Protecting and consuming services with OAuth
- Final thoughts

What is OpenSSO?

Web application security framework and service

Authentication, authorization, auditing, single sign-on, federation (e.g. SAML 2.0), web services, REST

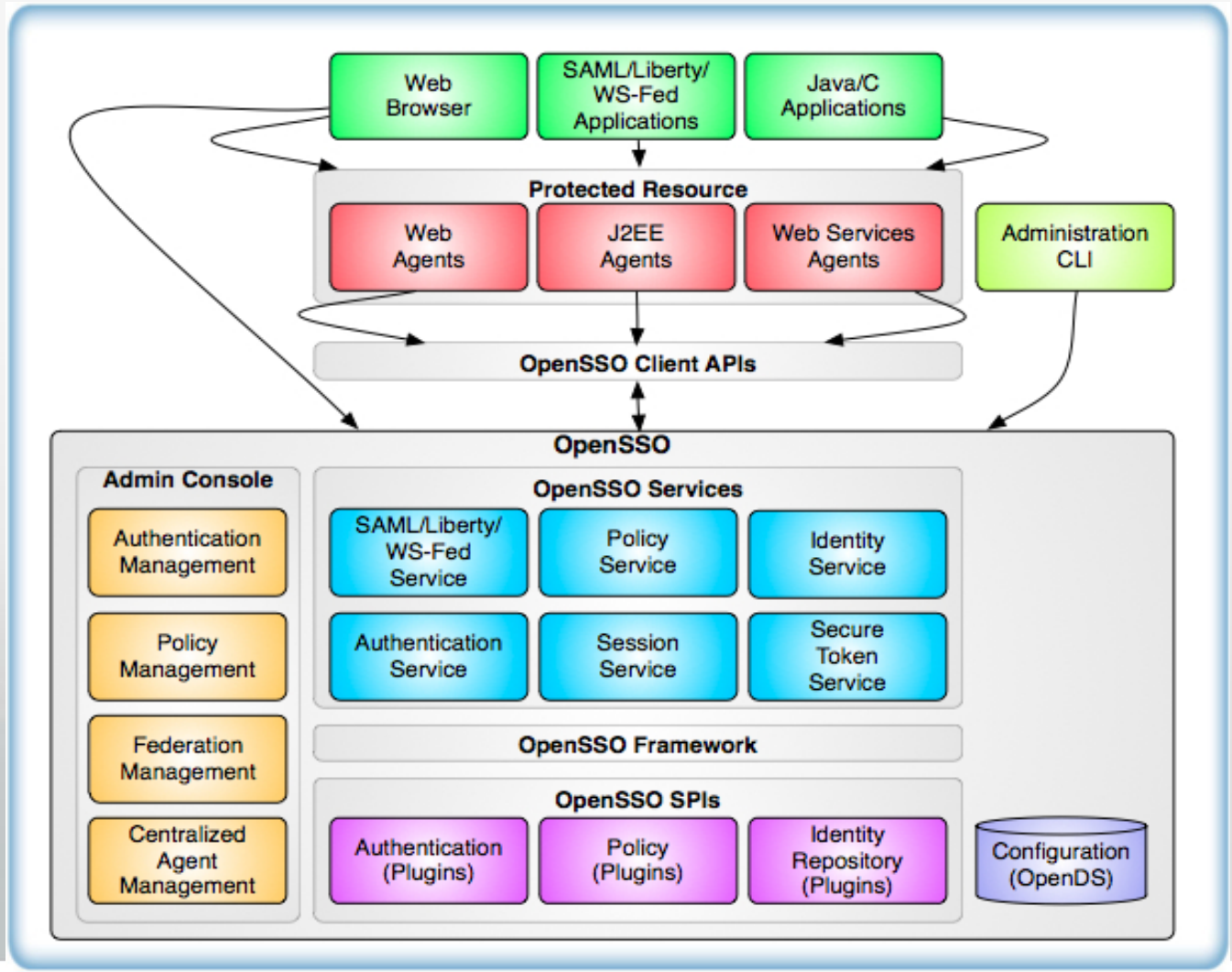
Handles security, you focus on application logic

Used by large companies and small websites

Open source and open community

http://www.opensso.org

OpenSSO architecture



Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

OpenSSO simple REST services APIs

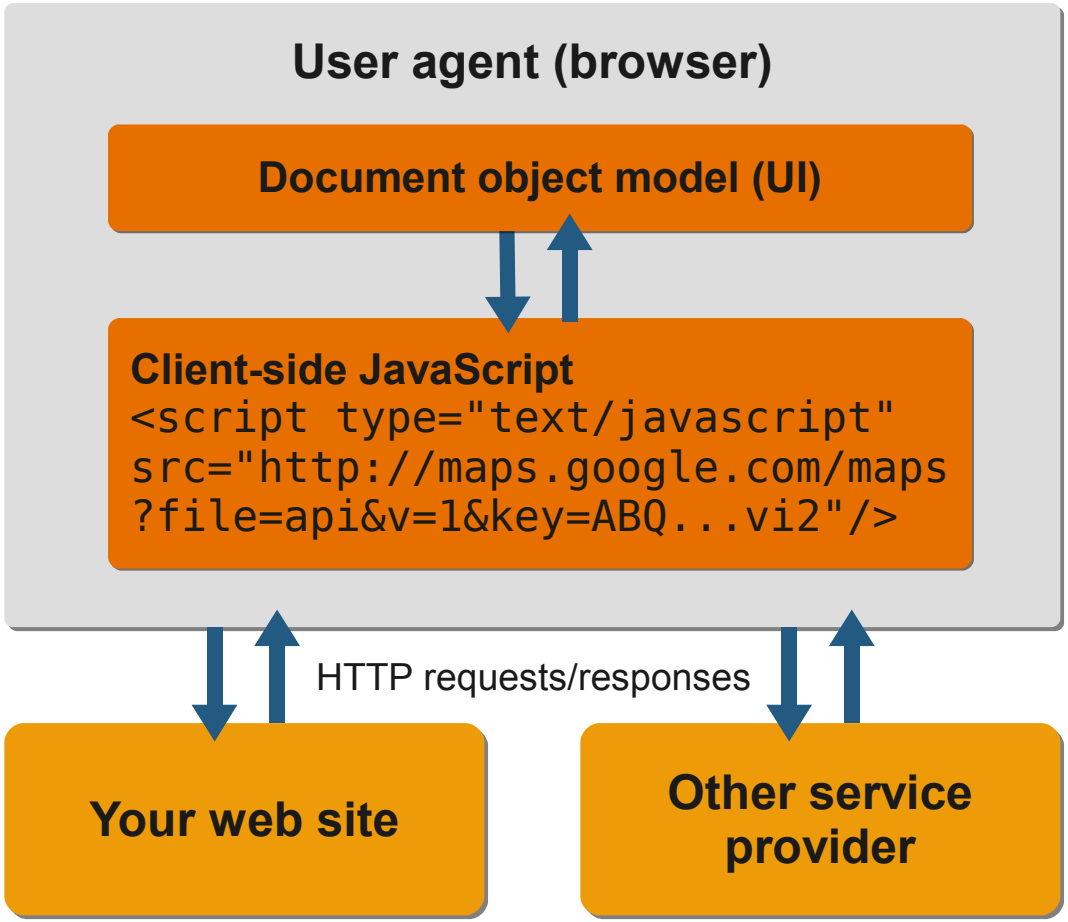
OAuth overview

Protecting and consuming services with OAuth

Final thoughts

Not in scope: client-side mashups

- Browser security model is king
- JavaScript libraries and widgets
- Tokens used in client
  - Example: maps
- Not all HTTP URLs are REST services



In scope: app-to-app REST services

Different levels of security

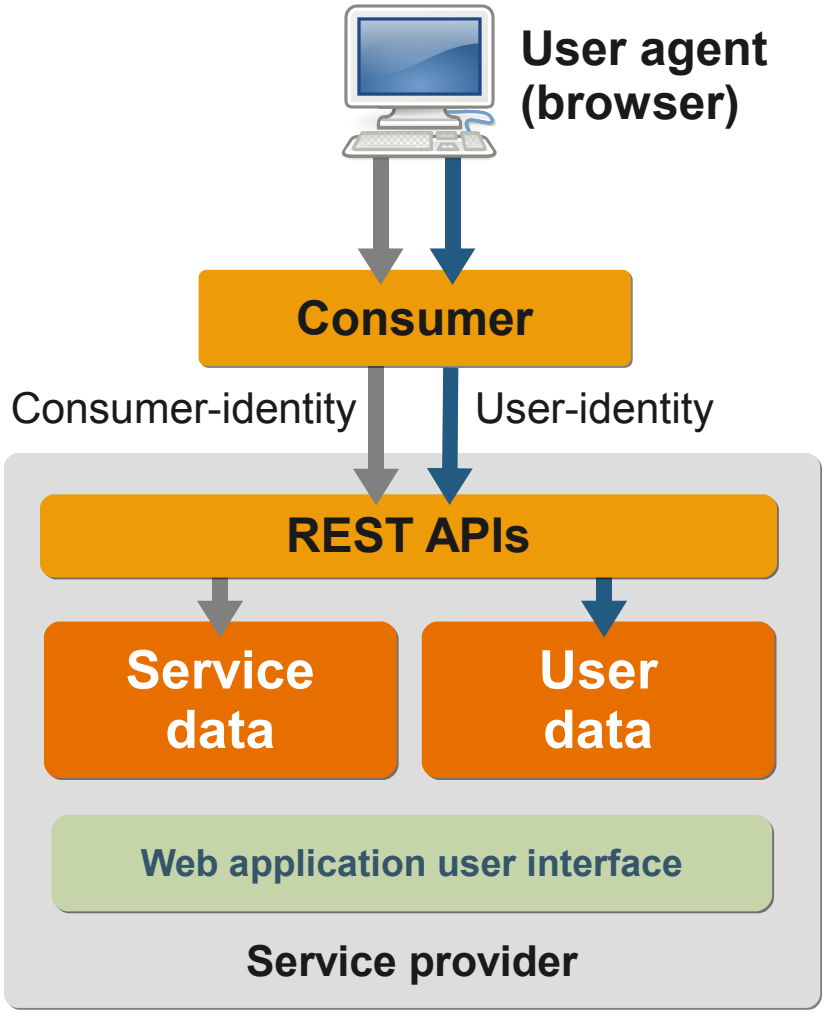
Tracking, metering, blog comments, semi-shared data among friends, private health care data

Different resources, different requirements

User data a driving force

Username and password?

Delegation:allow sites on your behalf to access some of your data



Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

OpenSSO simple REST services APIs

OAuth overview

Protecting and consuming services with OAuth

Final thoughts

REST architectural style

Resources: Identified by a URI

Methods: GET, POST, PUT, DELETE

Representations: Embody state

RPC vs. REST

- Few endpoints, many methods
- Many resources, few fixed methods

```
http://movies.example.com/access?type=movie&title=rocky-17
```

```
http://movies.example.com/movie/rocky-17
```

HTTP overview

Client and server, client can be a browser or another application

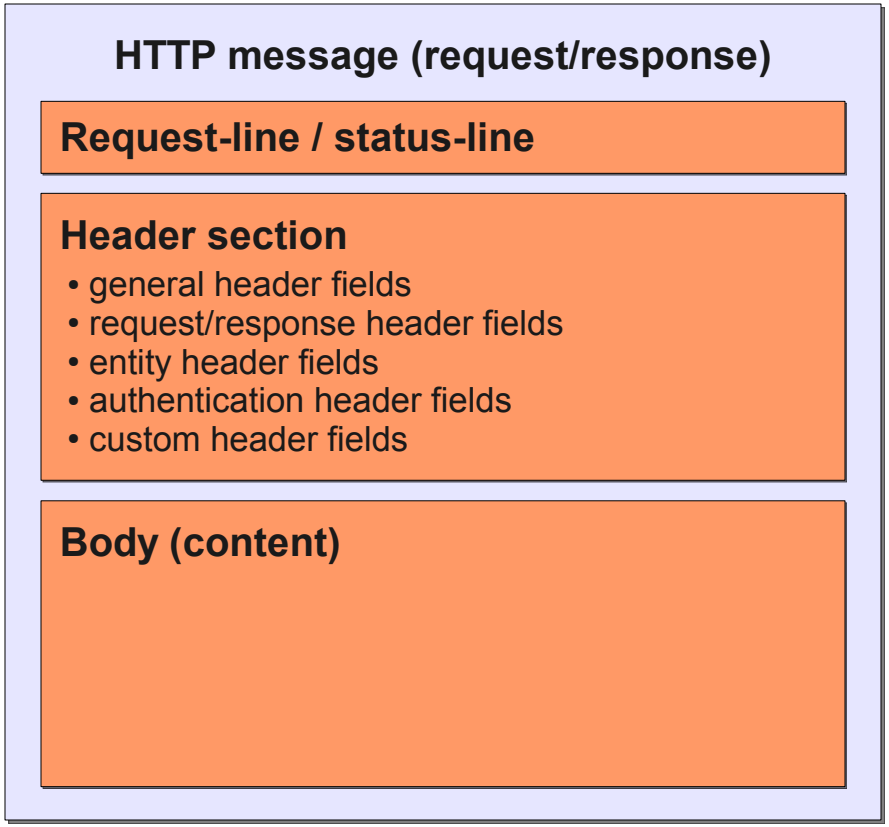
Requests and responses

Methods

Headers

HTTP status codes

200, 302, 400, 401, 403, 404



HTTP request/response sample

```
GET /test?foo=bar HTTP/1.1
Host: www.example.com
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
…
```

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 03 Jun 2009 23:11:55 GMT
Content-Type: text/plain; charset=UTF-8
…

Foo is "bar"
```

Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

OpenSSO simple REST services APIs

OAuth overview

Protecting and consuming services with OAuth

Final thoughts

REST security building blocks

Sender: add security into HTTP message, as header or URL query parameter

Receiver: pull security info out of HTTP message

- `request.getHeader("Authorization")`

Security information

  - Example: token, digital signature

Secure channel for messages with HTTPS

  - Point-to-point, transport layer
  - Confidentiality and message integrity

Tokens, Tokens, Tokens?

Keep in mind

AQIC5wM2LY4SfcyqRg7a/151nDNvPySQyCoI6ZwGkrfc@AAJTSQAC

- How you get it: registration & inputs
- Runtime tokens
- What tokens looks like
- Who the token represents
- Permissions, impersonation/delegation
- Exposing tokens
  - Characteristics, duration, one time, persistence
- How to transmit tokens: headers, URLs, cookies

Message-level security

- Application embeds security into the message
  - Example: digital signature added as a header on a an HTTP request
- Digital signatures added to message
  - Message integrity
  - Sender generates a signature on message
  - Add the signature as a header to your HTTP request, and request is considered signed
  - Receiver verifies
- Often still sent over HTTPS

Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

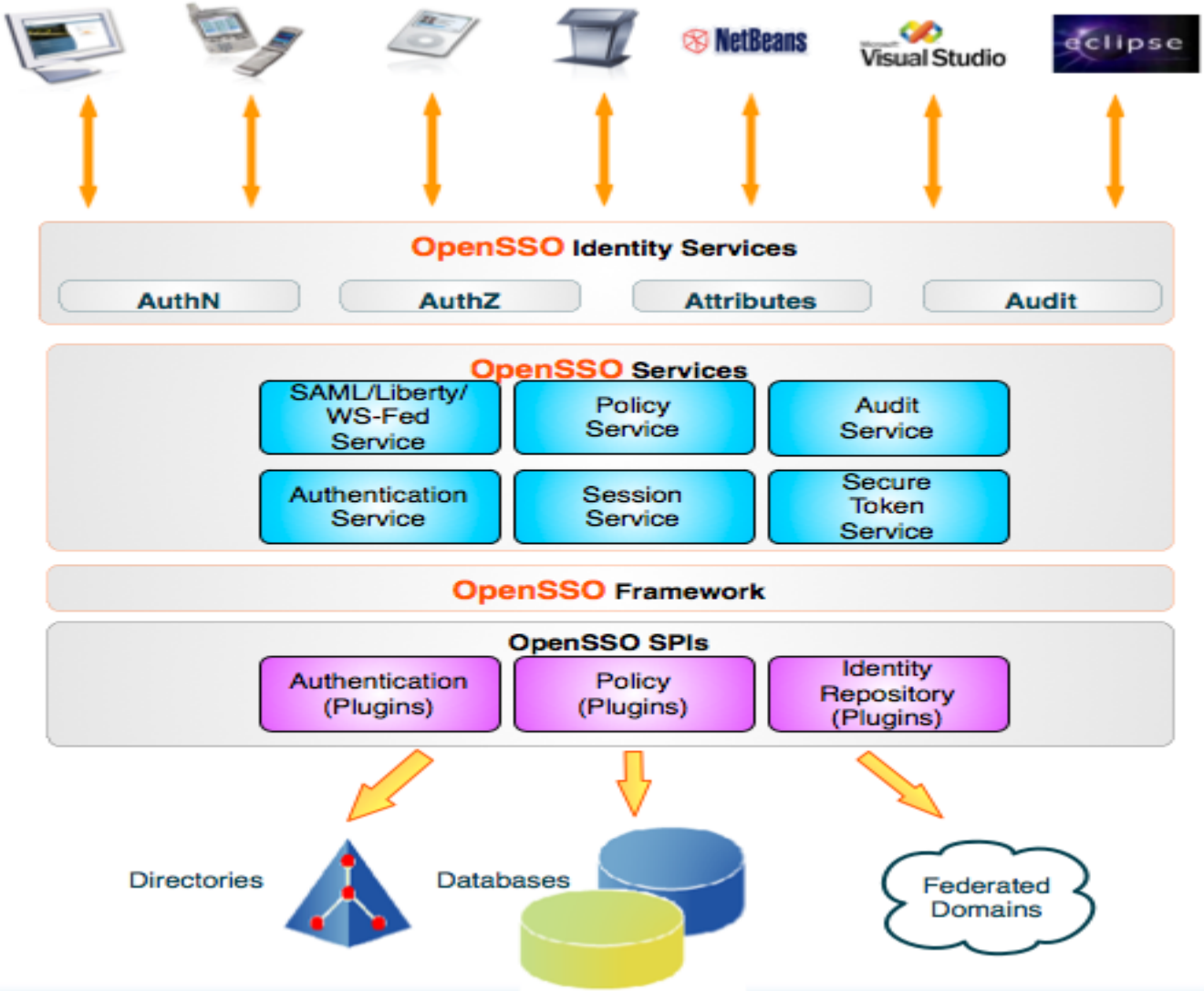
OpenSSO simple REST services APIs

OAuth overview

Protecting and consuming services with OAuth

Final thoughts

Identity services architecture



Simple REST-like APIs for OpenSSO

First generation of services

Authentication

Verification of user credentials

```
POST .../authenticate?
username=demo&password=demo
```

Authorization

Permission for user to access protected resource

```
GET .../authorize?token=aaa&
resource=bbb&action=ccc...
```

Attributes

Obtain attributes of users

```
GET .../attributes?
token=aaa&attributes_names=cn
```

Audit log

Perform log & audit operations

```
POST .../log?appid=aaa&
subjectid=bbb=cn&logname=...
```

Authentication (text/plain)

Request

```
POST /opensso/identity/authenticate
```

Form data: `username=amadmin&password=11111111`

Response

```
Content-Type: text/plain;charset=UTF-8

token.id=AQIC5wM2LY4SfcwKyiCJAv2xL/fJiTfKaoUl+IKIfD1Ni
Wg=@AAJTSQACMDE=#
```

Authentication (text/xml)

Request

```
POST /opensso/identity/xml/authenticate
```

Form data: `username=amadmin&password=11111111`

Response

```
Content-Type: text/xml;charset=UTF-8

<token id="AQIC5wM2LY4SfcyqRg7a/151nDNvPySQyCIbToI6ZwG
krfc=@AAJTSQACMDE=#"/>
```

Attributes (text/plain)

Request

```
GET  /opensso/identity/attributes

Cookie: iPlanetDirectoryPro="AQIC5wM2LY4SfcxL8yo/RS
        VzGbQYpvlz50VZpE2KI1yjNNg=@AAJTSQACMDE=#"
```

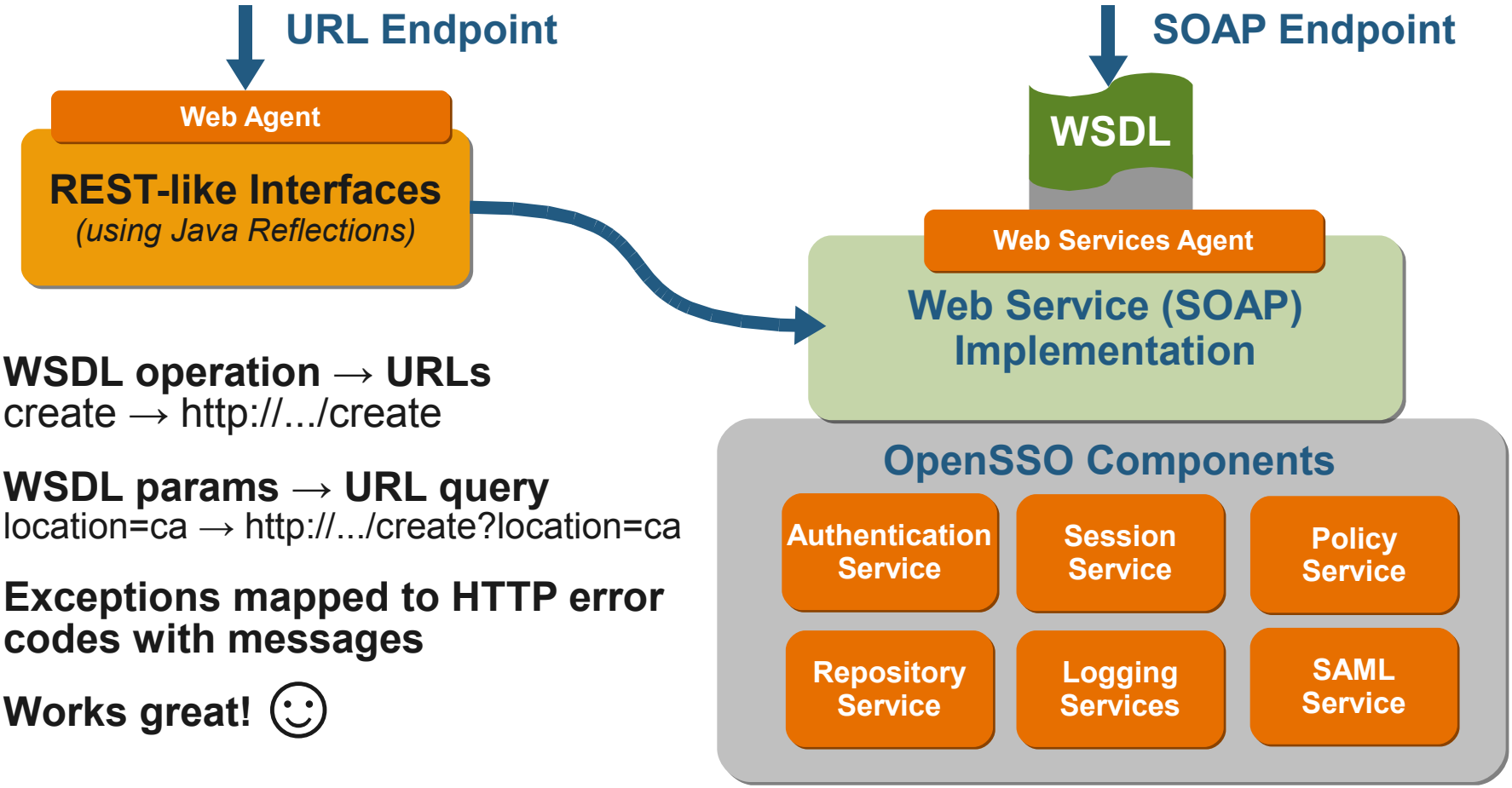
Response

```
Content-Type: text/plain;charset=UTF-8

userdetails.attribute.name=sunidentitymsisdnnumber
userdetails.attribute.name=mail
userdetails.attribute.name=sn
userdetails.attribute.value=amAdmin
userdetails.attribute.name=givenname
userdetails.attribute.value=amAdmin
```

Jersey Client sample to authenticate

```
Client client = Client.create();

WebResource resource = client.resource(
    "http://example.com/opensso/identity/authenticate");

MultivaluedMap form = new MultivaluedMapImpl();
form.add("username", "amadmin");
form.add("password", "11111111");

String response = webResource.type(
    "application/x-www-form-urlencoded")
    .post(String.class, form);
```

Design and implementation
First generation of services



- **WSDL operation → URLs**
  create → http://.../create
- **WSDL params → URL query**
  location=ca → http://.../create?location=ca
- **Exceptions mapped to HTTP error codes with messages**
- **Works great!** ☺

Pros & cons of using identity services

First generation of services

Programming language agnostic. OpenSSO is not restricted to Java and C languages.

Need for client SDK?

Caching? How can consumer site cache the authorization decisions, user attributes, etc, from OpenSSO server? Maybe a need for SDK.

Exceptions? Mapping of HTTP error codes and passing of error messages.

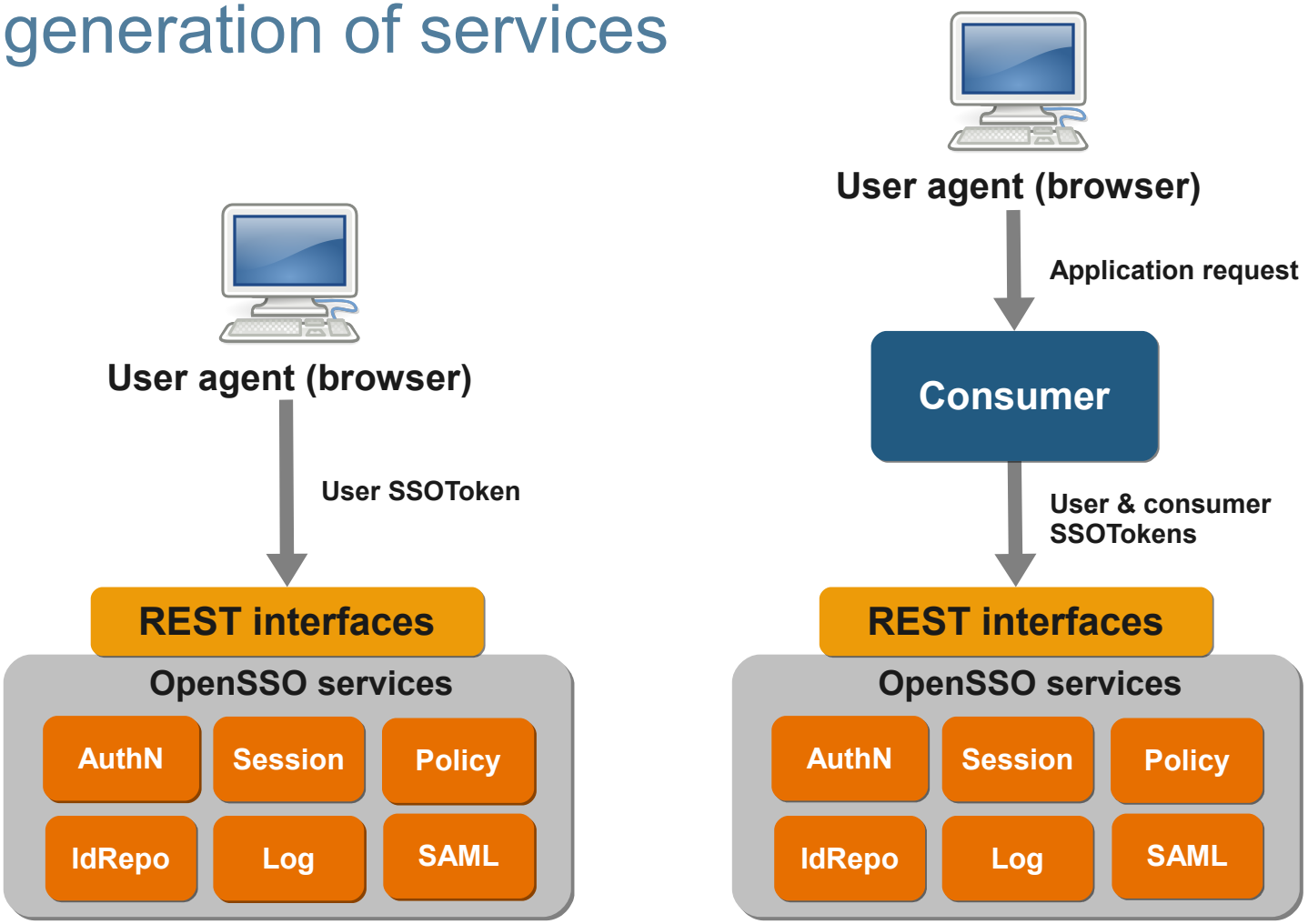
Security? Communication between client and OpenSSO servers & rouge clients.

OpenSSO REST security
First generation of services

- Authentication & authorization of callers to REST URLs
  - Course-grained policy enforcement based on URL
- Fine-grained authorization within the application logic
  - Examples: access to attributes, ability to log, etc.
- Authentication Interface should be unprotected
- Session established and maintained after authentication
  - SSOToken: random string usually stored as cookie
- SSOToken passed in each request
  - As either cookie or query parameter
- Key parameters passed as query parameters

OpenSSO REST security

First generation of services



Lessons learned

- Imperfect RESTful APIs
  - Current application not easy to convert to URL resources like REST
- Message authentication
- Requires user presence
- Consumer could masquerade as the user
- Token management
- Still useful
  - Allow other languages
  - A step toward more RESTful

Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

OpenSSO simple REST services APIs

OAuth overview

Protecting and consuming services with OAuth

Final thoughts

OAuth overview

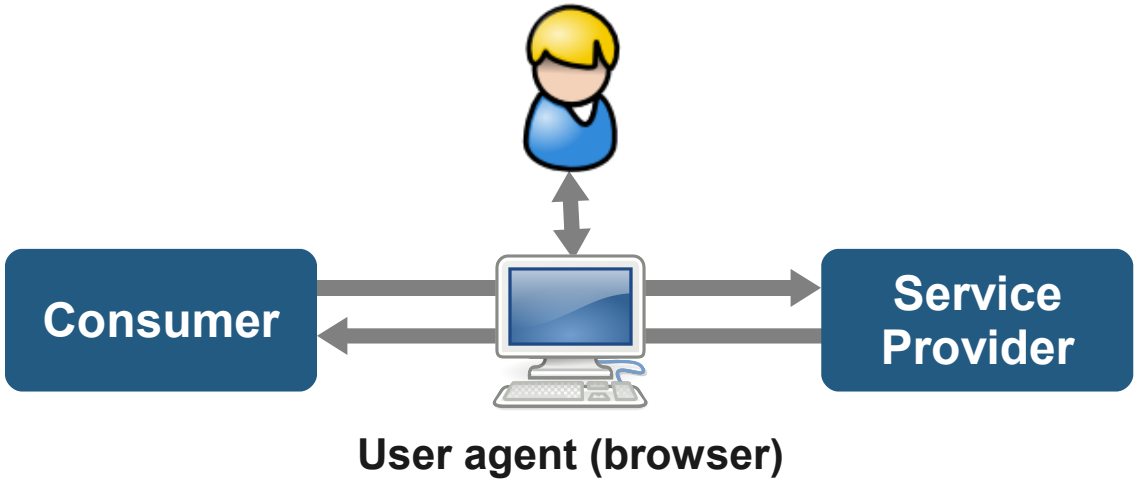
Users securely share their resources in one service with another service without exposing credentials.

Prototypical use case: user shares images from an image gallery with a photo printing service.

Once user brokers issuance of token, it can be used on an ongoing basis. Think: session keys for consumer applications.

Provides a very handy consumer authentication capability through the OAuth digital signature.

1 User brokers issuance of access token



User introduces service provider to consumer

Authorization performed through browser redirects

Standard user authentication with service provider

Access token is issued to consumer on behalf of user

❷ Consumer accesses resource directly

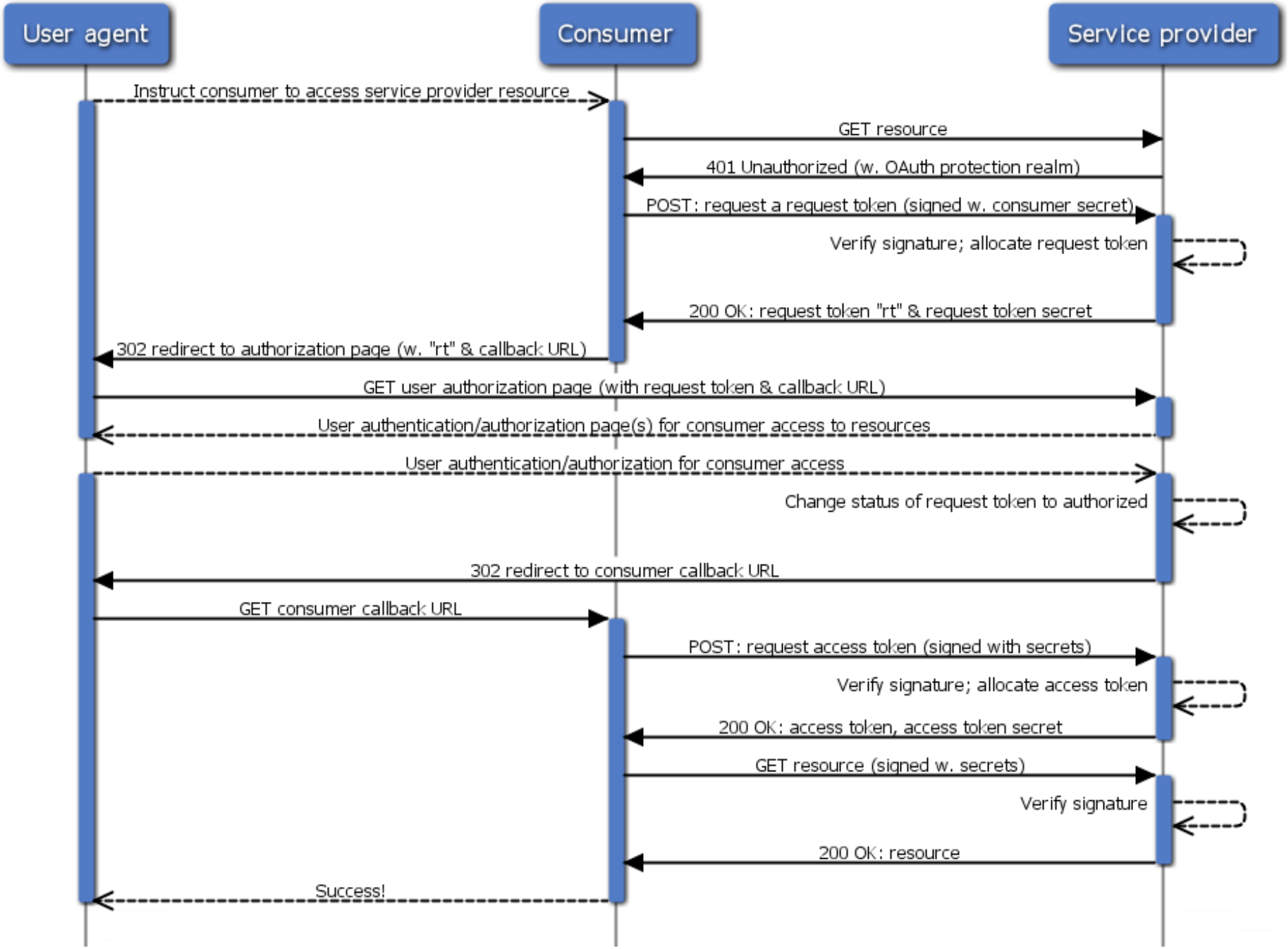
Consumer signs requests with access token secret

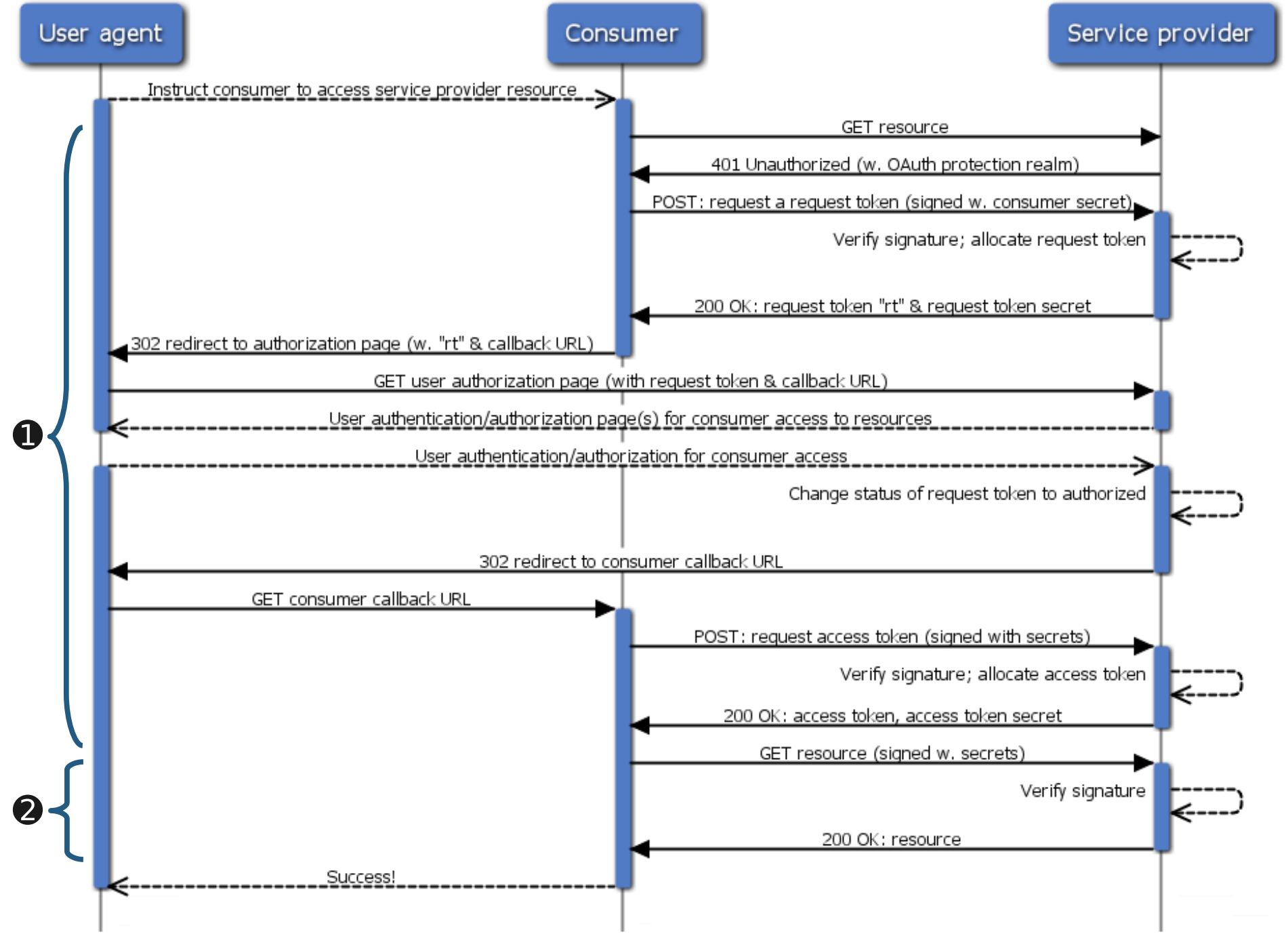
Service provider can enforce its own access controls

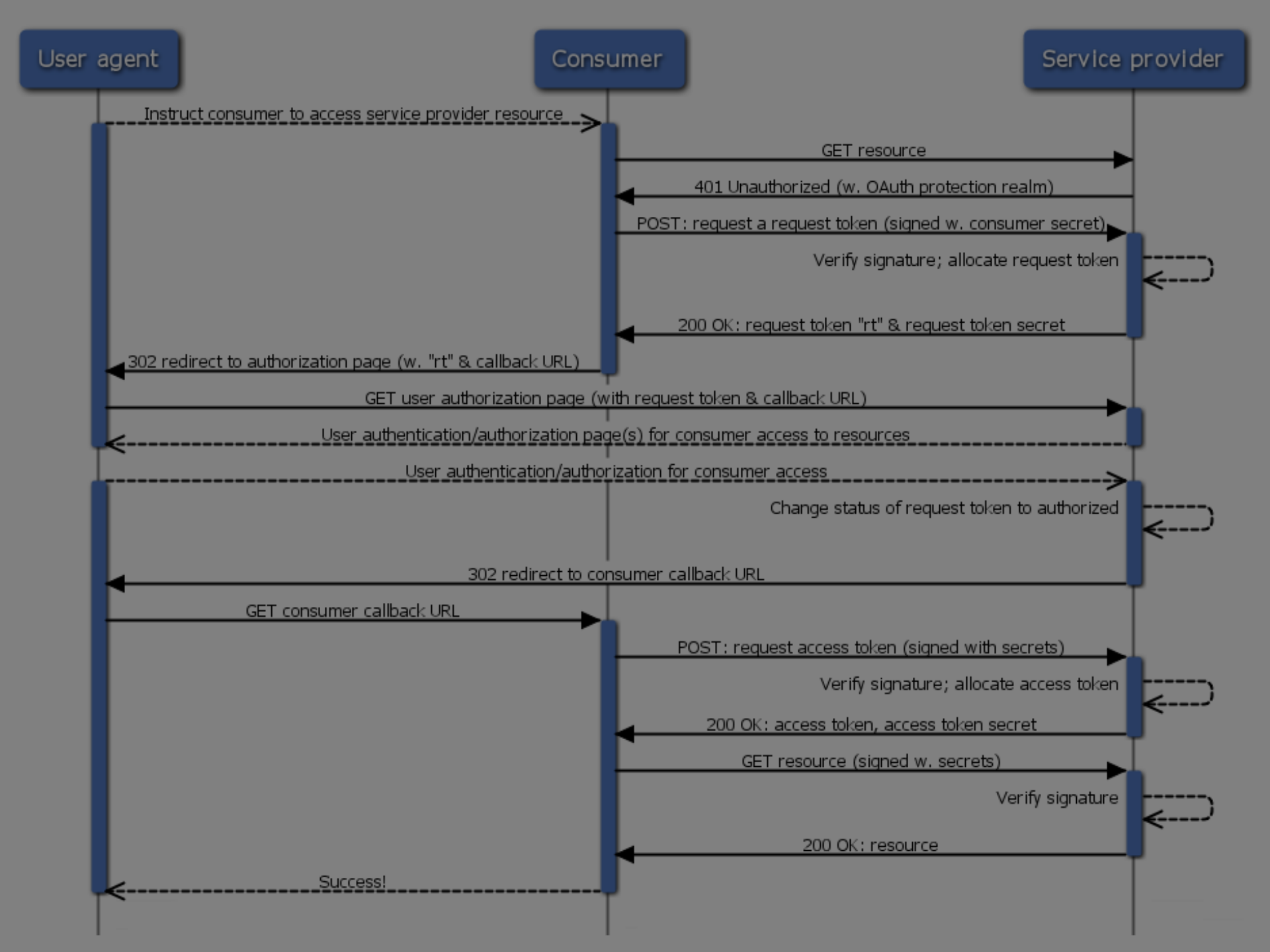
Doesn't require constant user presence

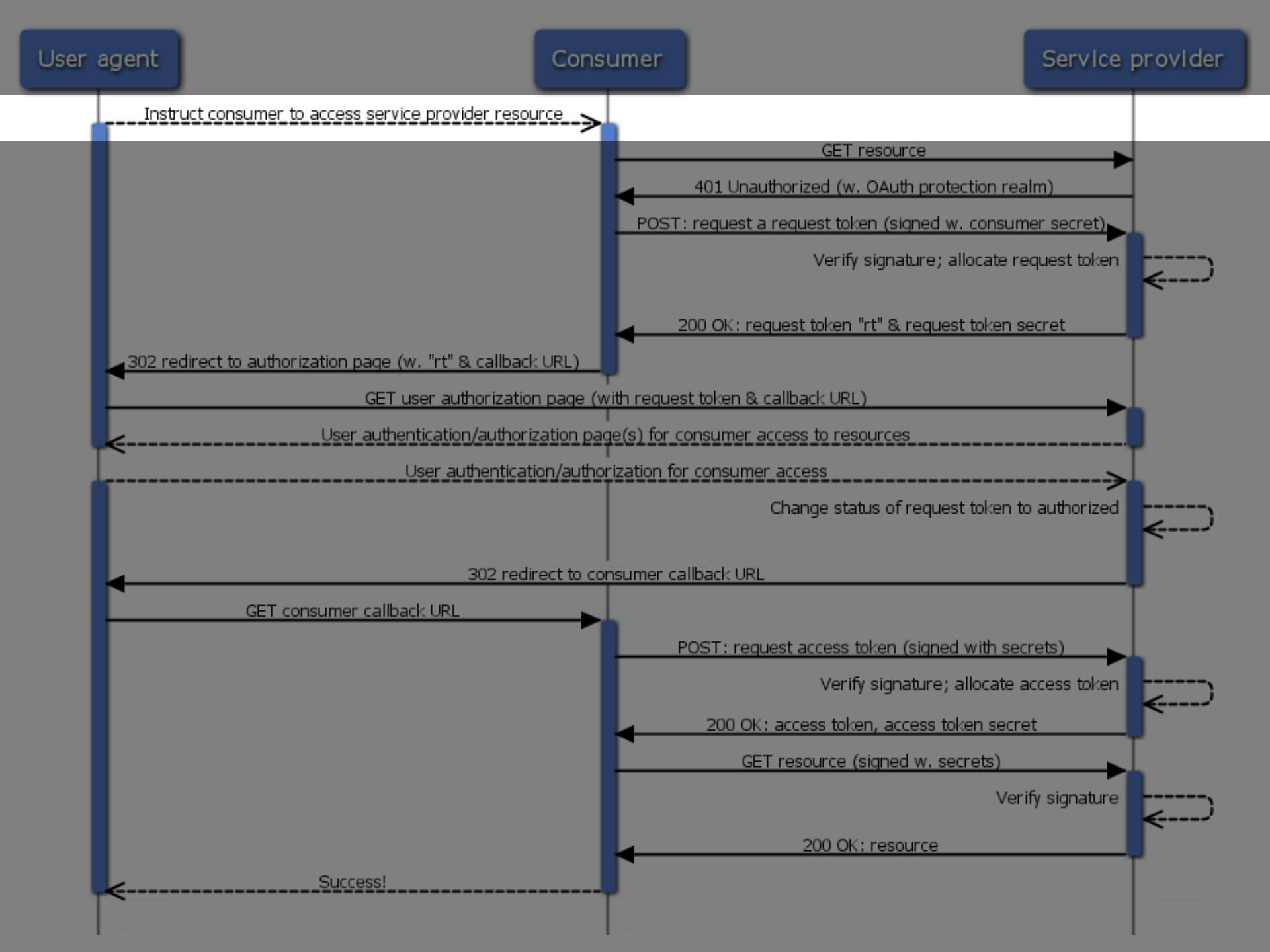
Detailed OAuth flow walkthrough

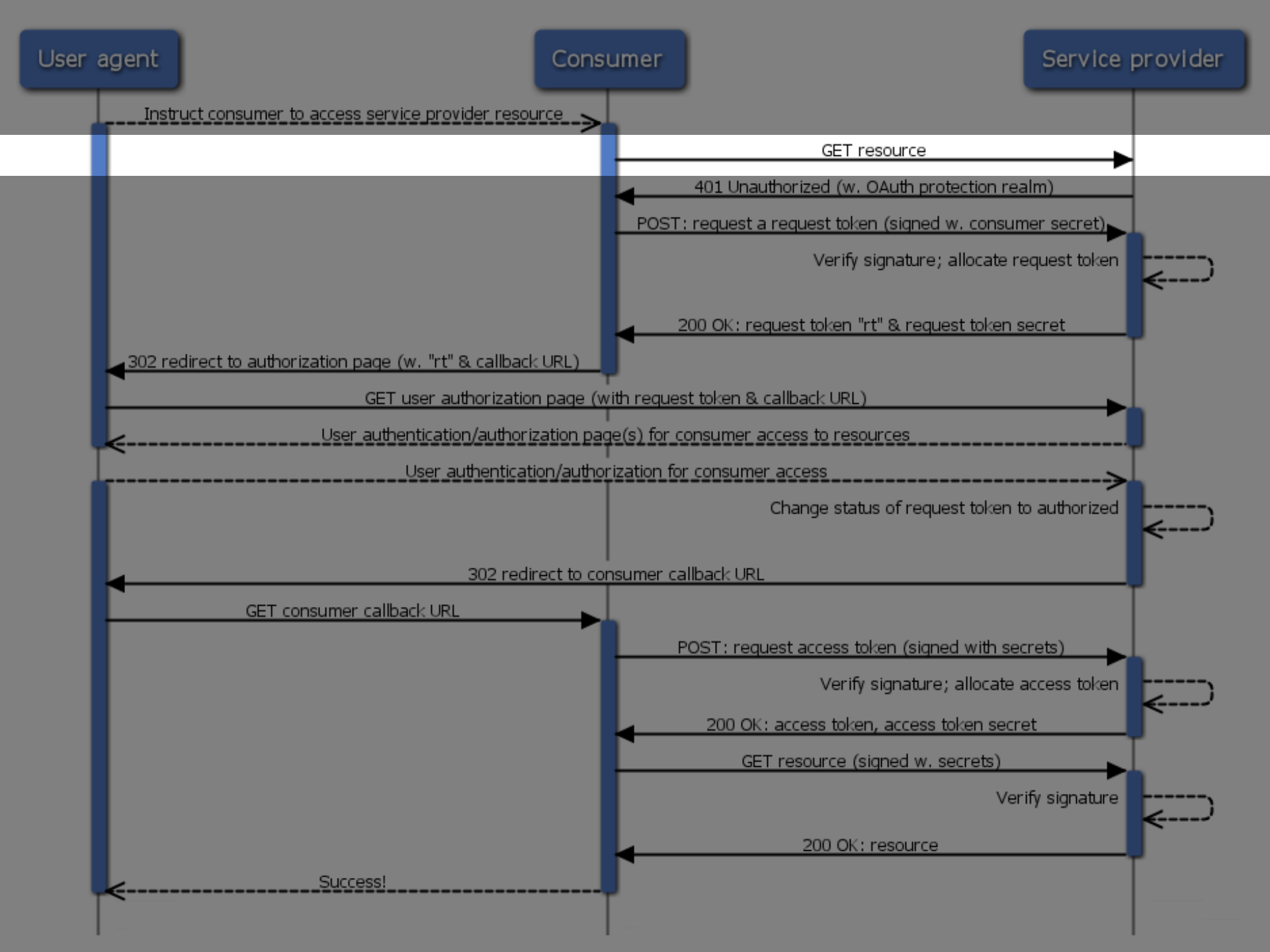
Let's take a look in more detail...

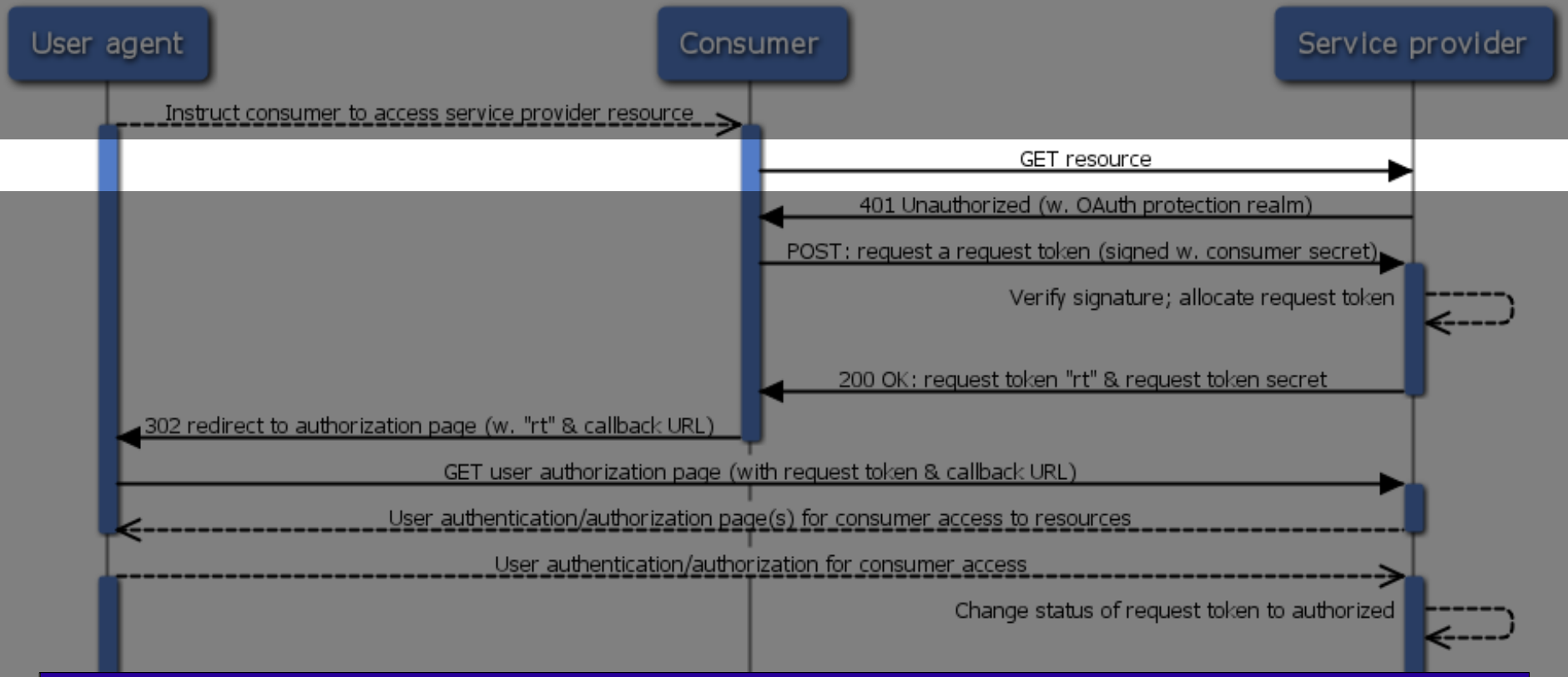

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!
1
2


User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

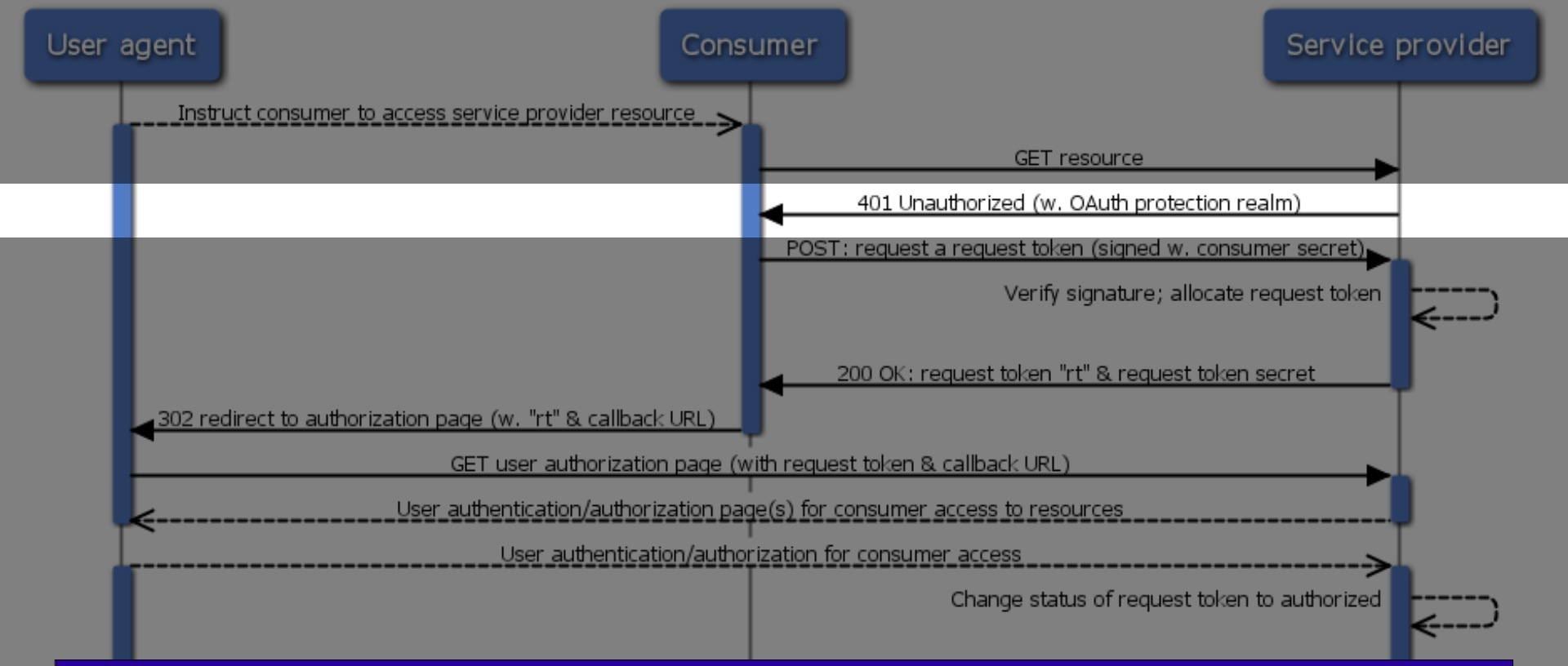

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!



Handy HTTP protocol analyzer

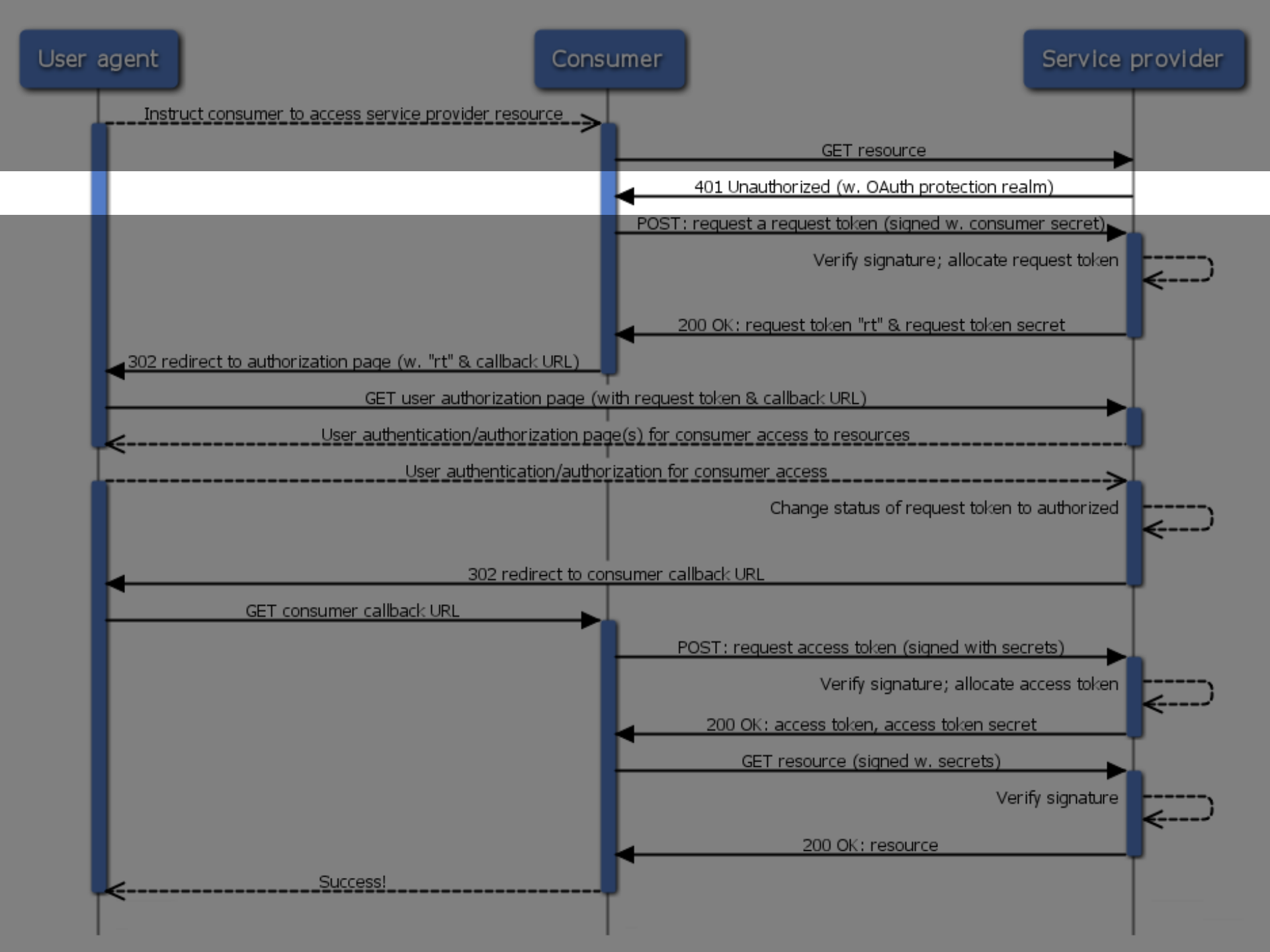
```
GET /photos?file=vacation.jpg&size=original HTTP/1.1
…
```

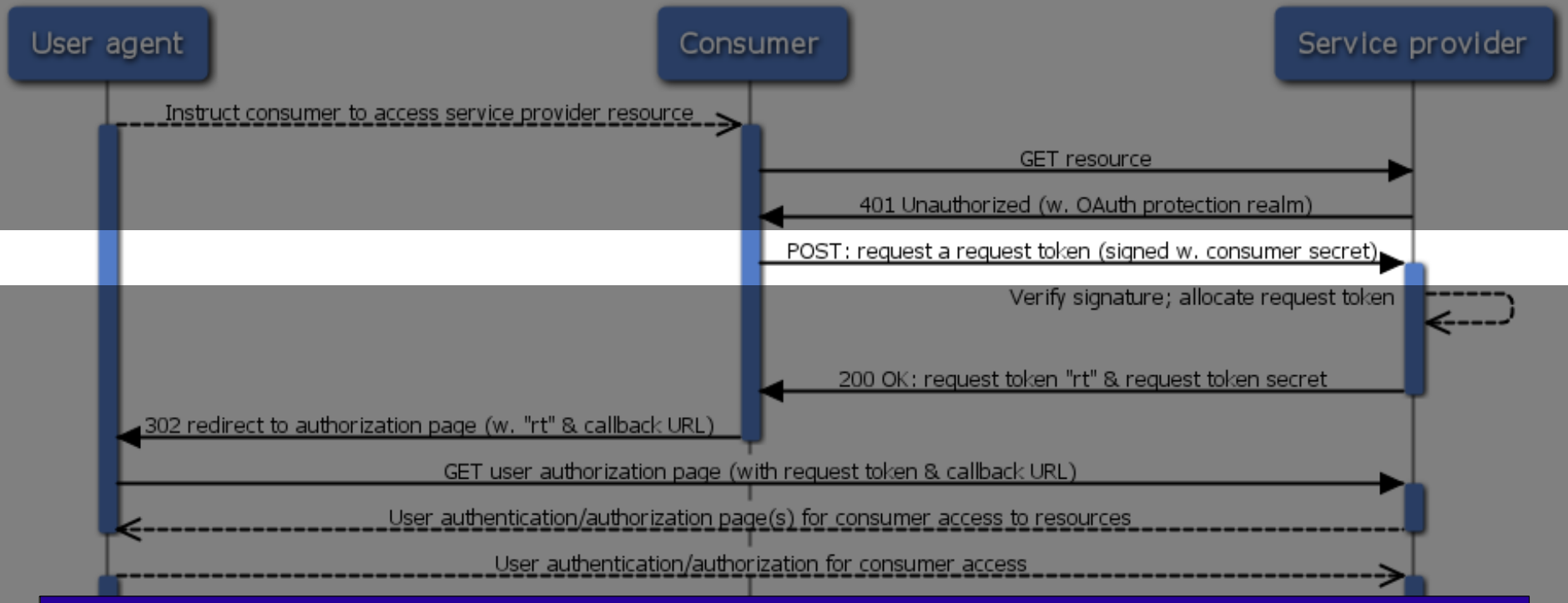


Handy HTTP protocol analyzer

```
GET /photos?file=vacation.jpg&size=original HTTP/1.1
…
```

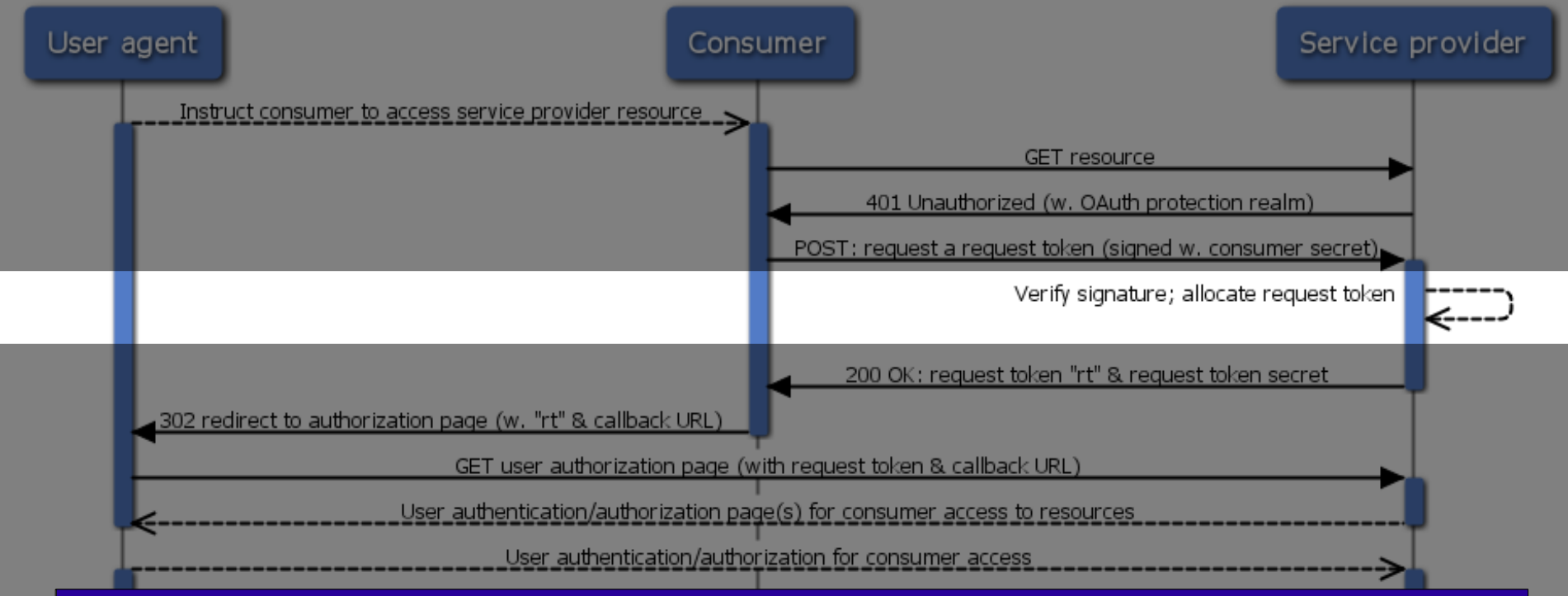
```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: OAuth realm="http://photos.example.net/"
…
```


User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!



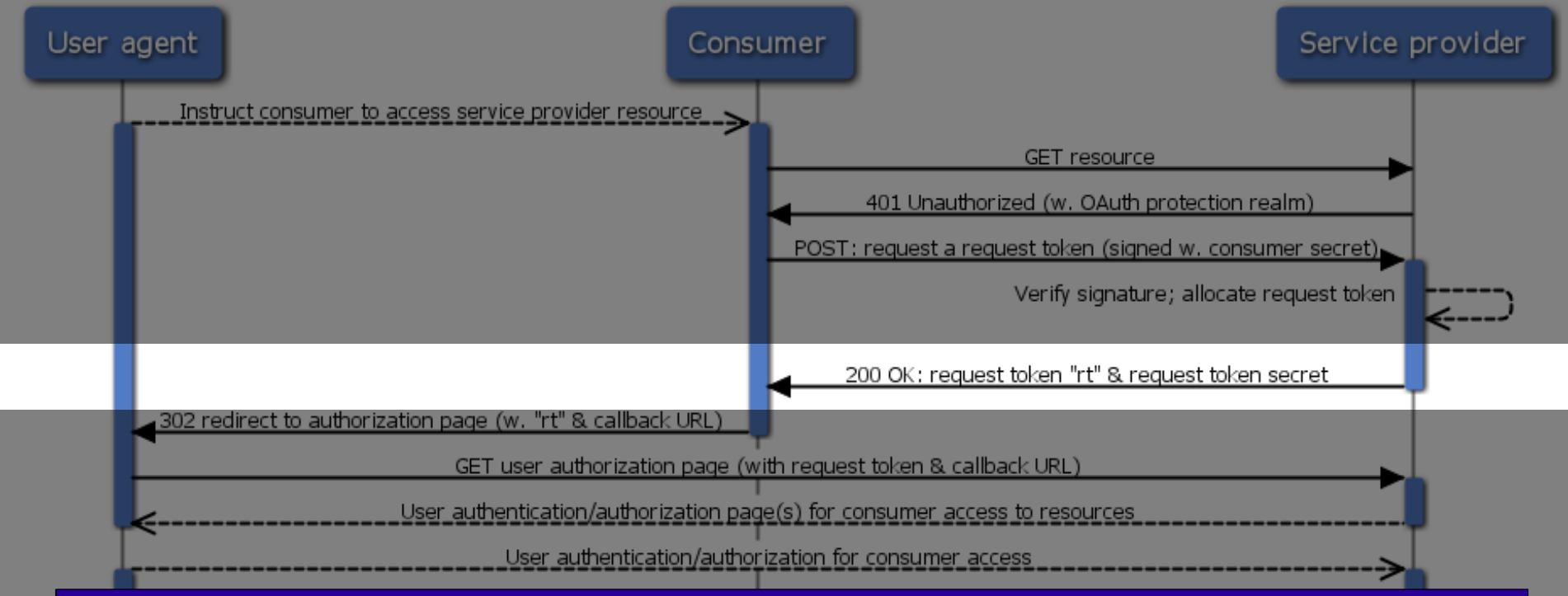
Handy HTTP protocol analyzer

```
POST /request_token?oauth_consumer_key=dpf43f3p2l4k3l03&
 oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26&
 oauth_timestamp=1191242090&oauth_nonce=hsu94j3884jdopsl&oauth_version=1.0 HTTP/1.1
...
```



Handy HTTP protocol analyzer

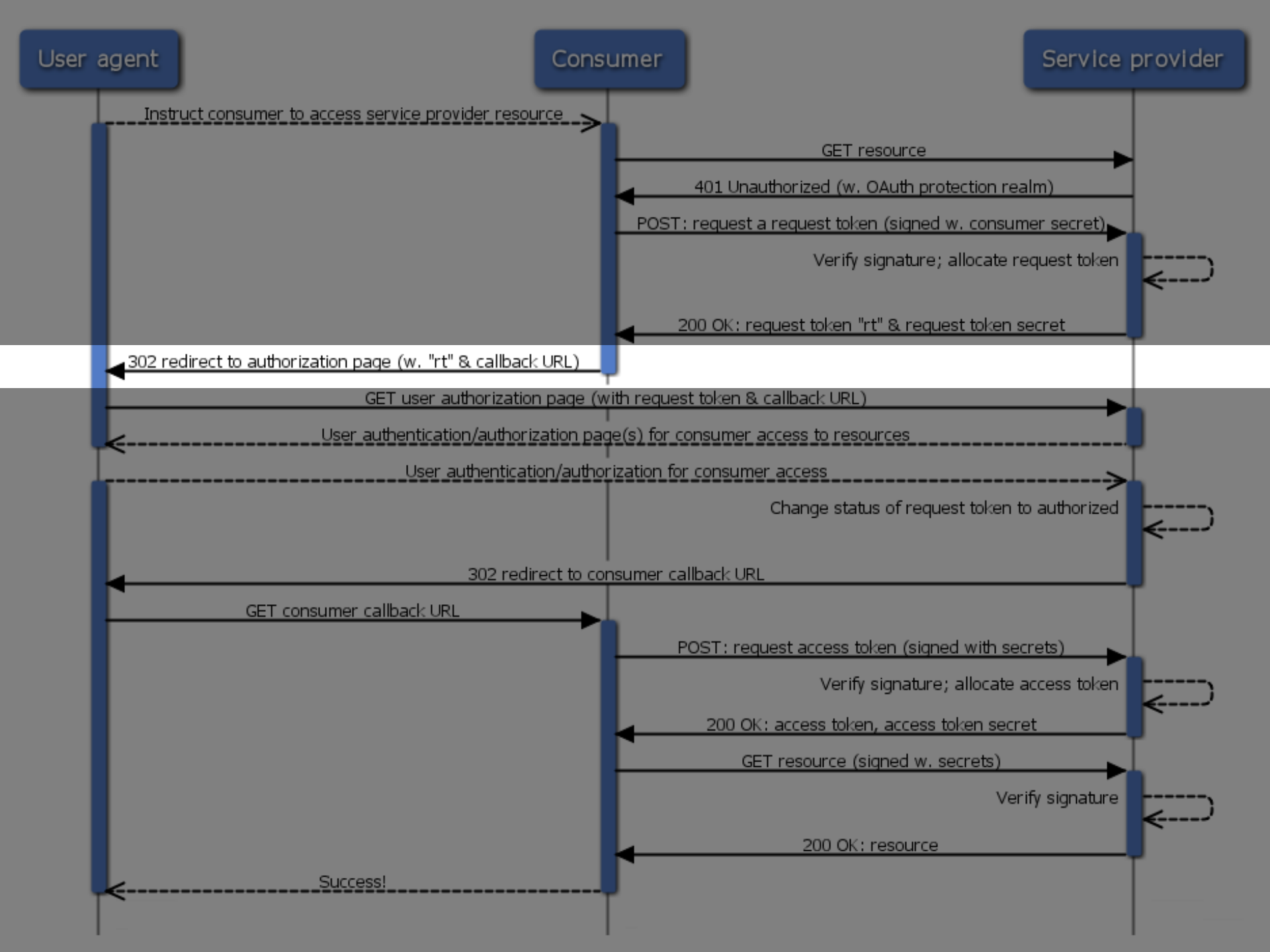
```
POST /request_token?oauth_consumer_key=dpf43f3p2l4k3l03&
 oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26&
 oauth_timestamp=1191242090&oauth_nonce=hsu94j3884jdopsl&oauth_version=1.0 HTTP/1.1
...
```

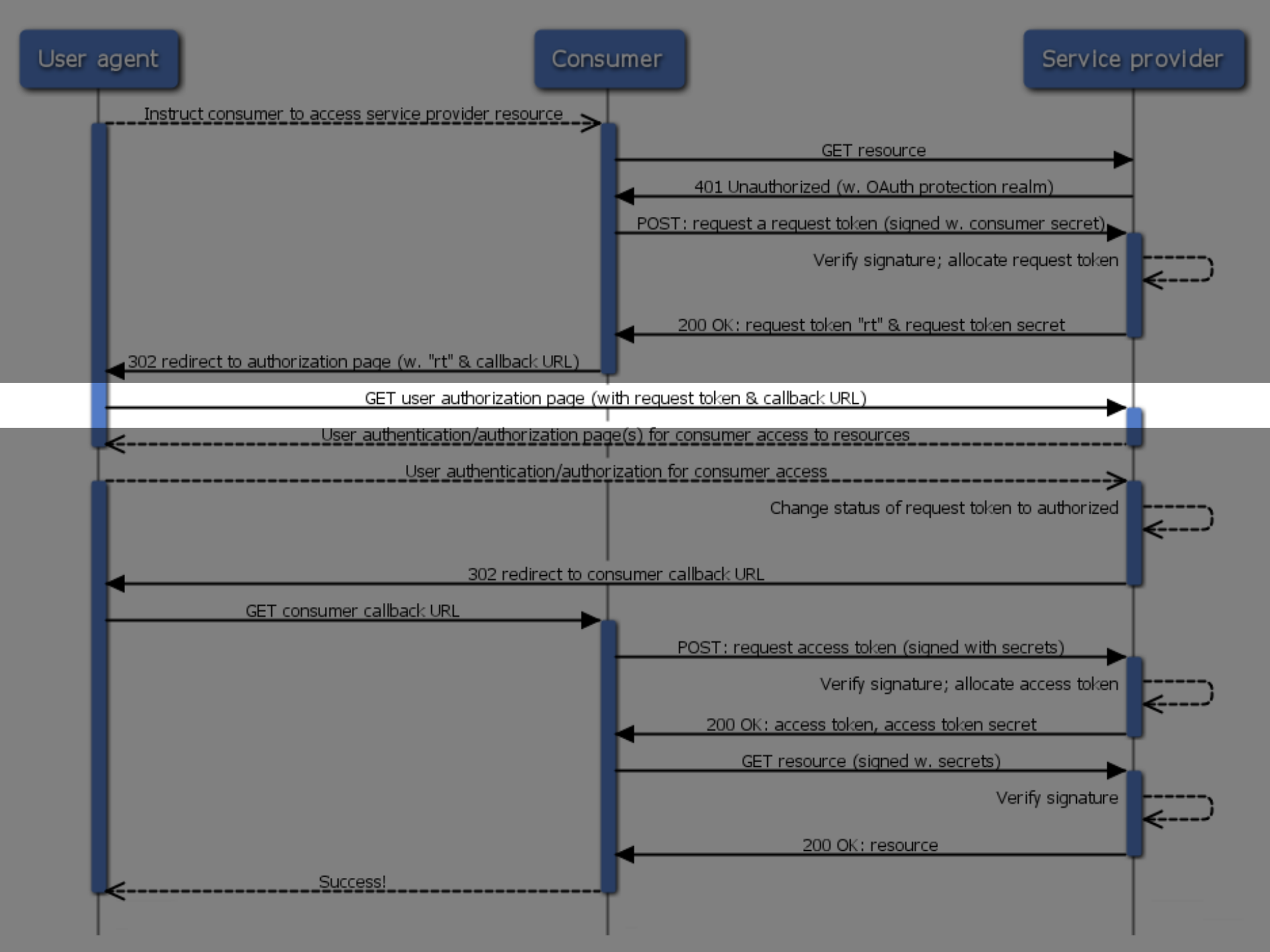


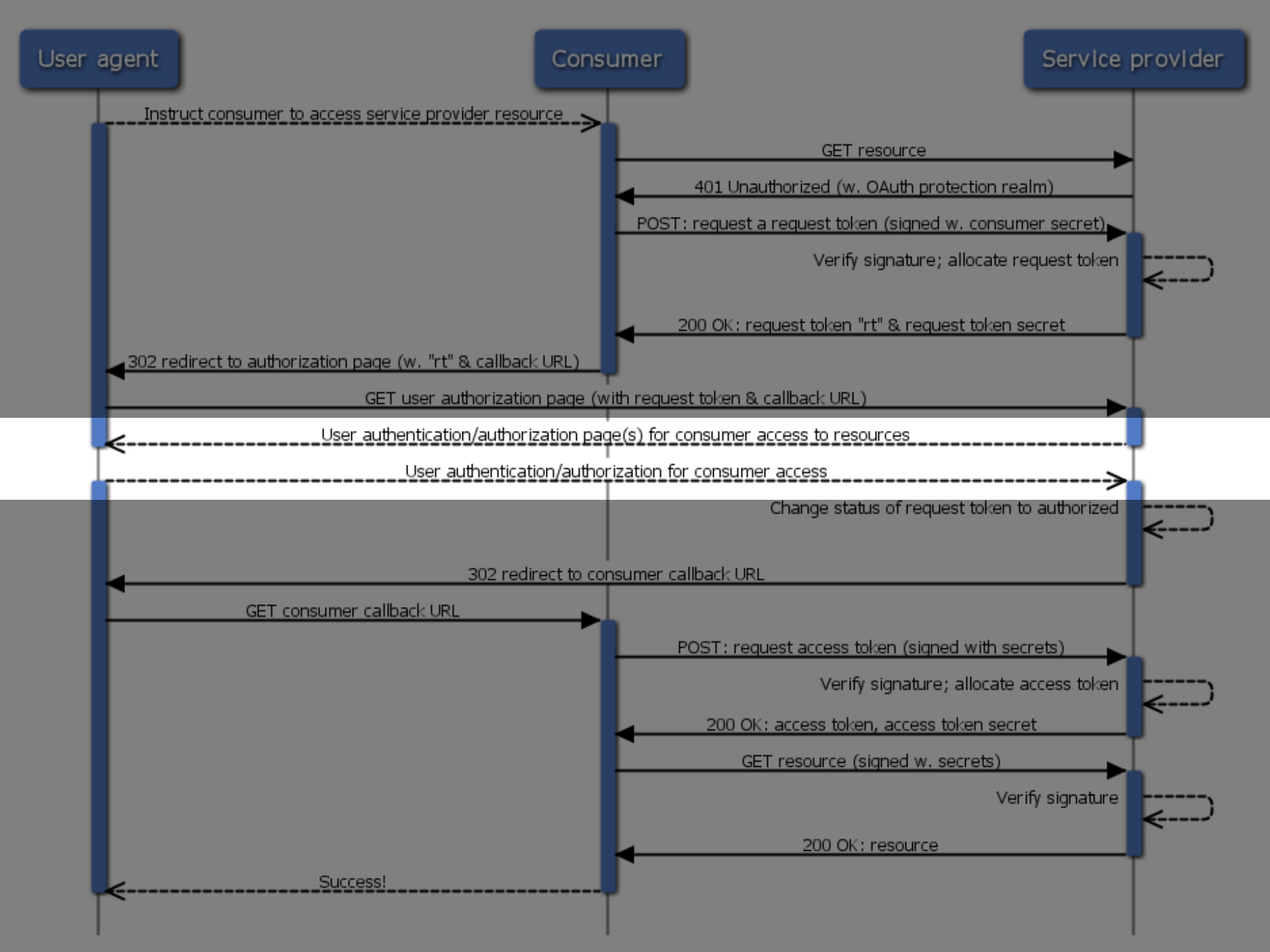
Handy HTTP protocol analyzer

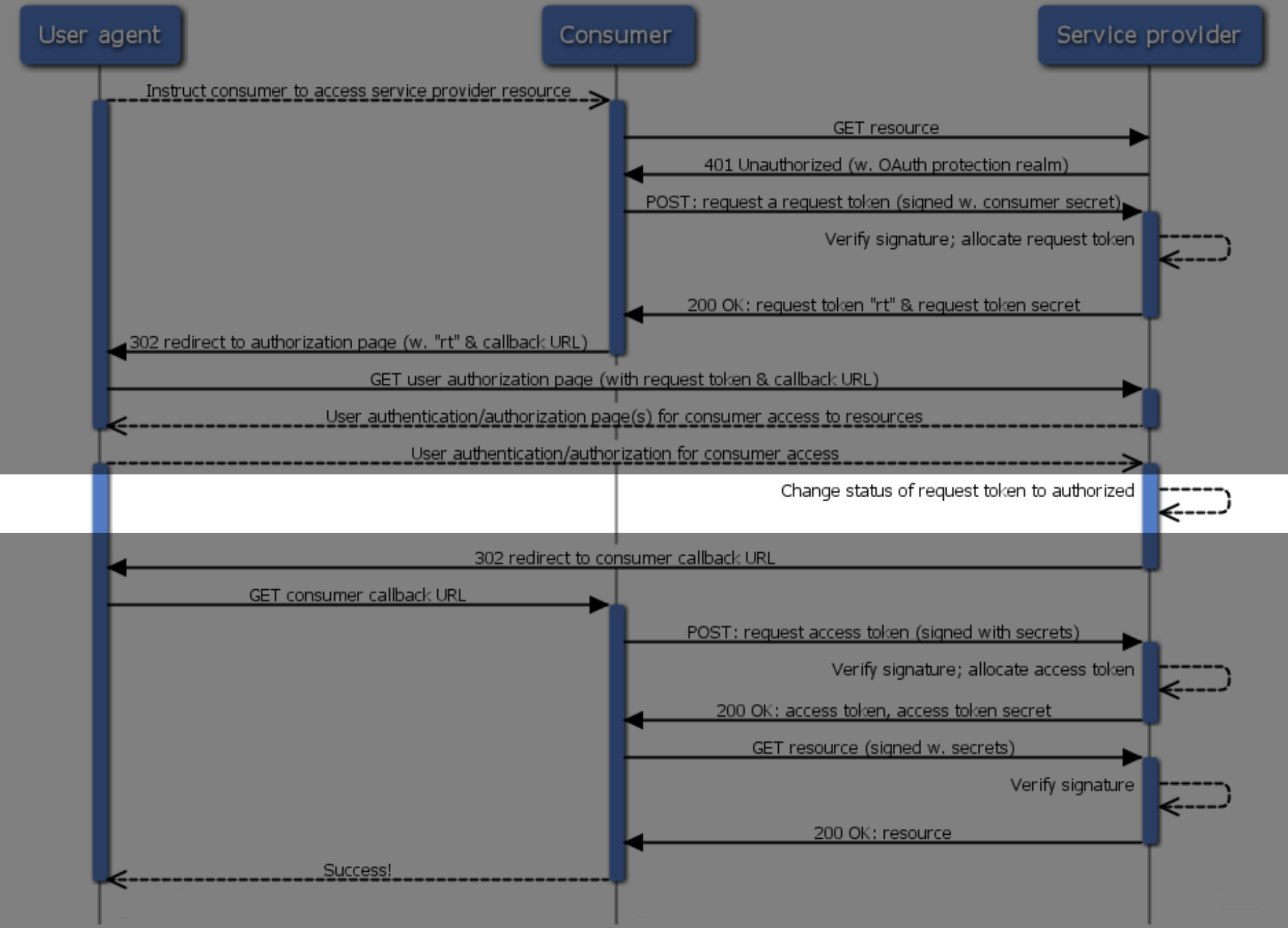
```
POST /request_token?oauth_consumer_key=dpf43f3p2l4k3l03&
 oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26&
 oauth_timestamp=1191242090&oauth_nonce=hsu94j3884jdopsl&oauth_version=1.0 HTTP/1.1
…
```

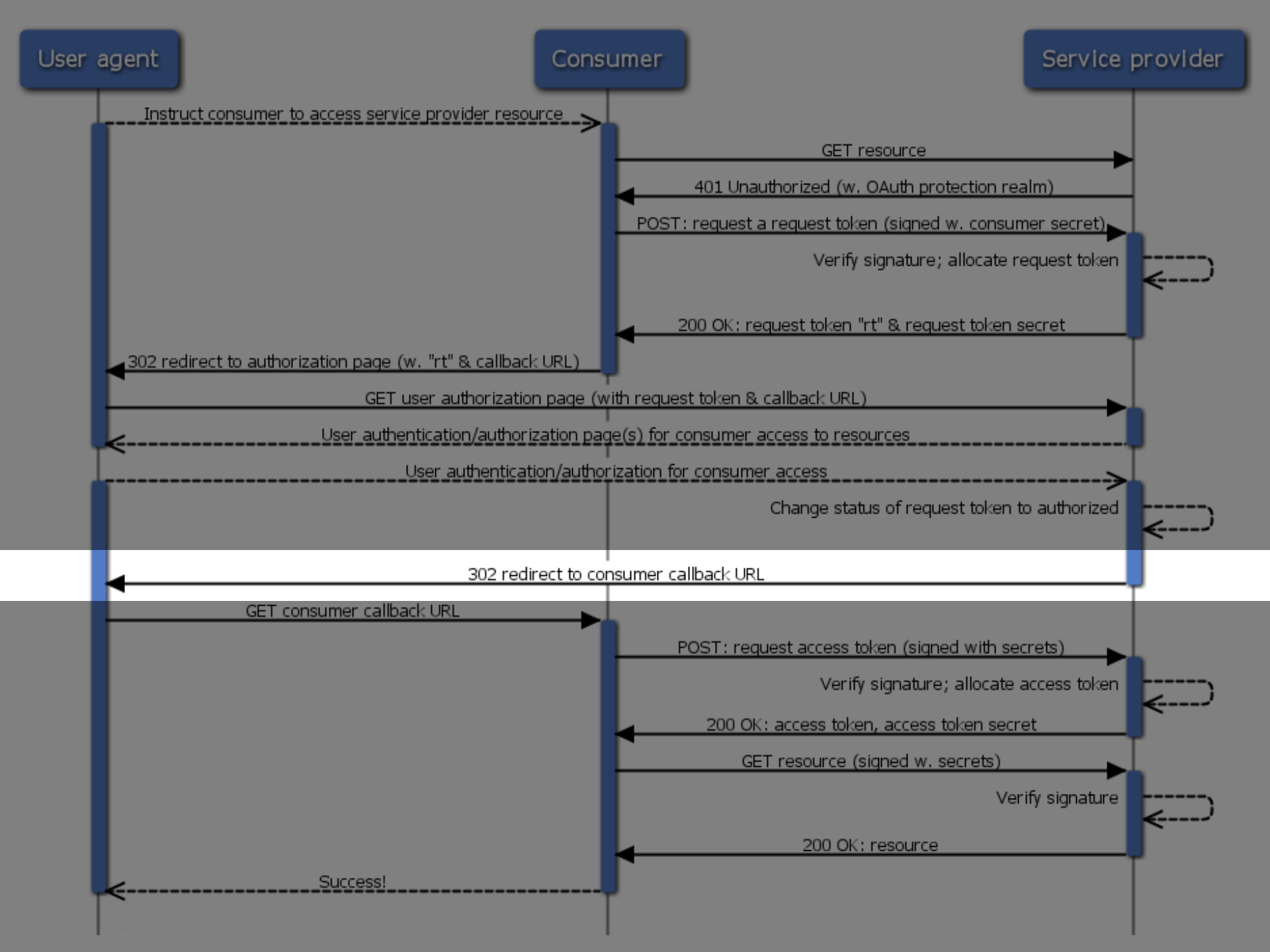
```
HTTP/1.1 200 OK
…

oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03
```

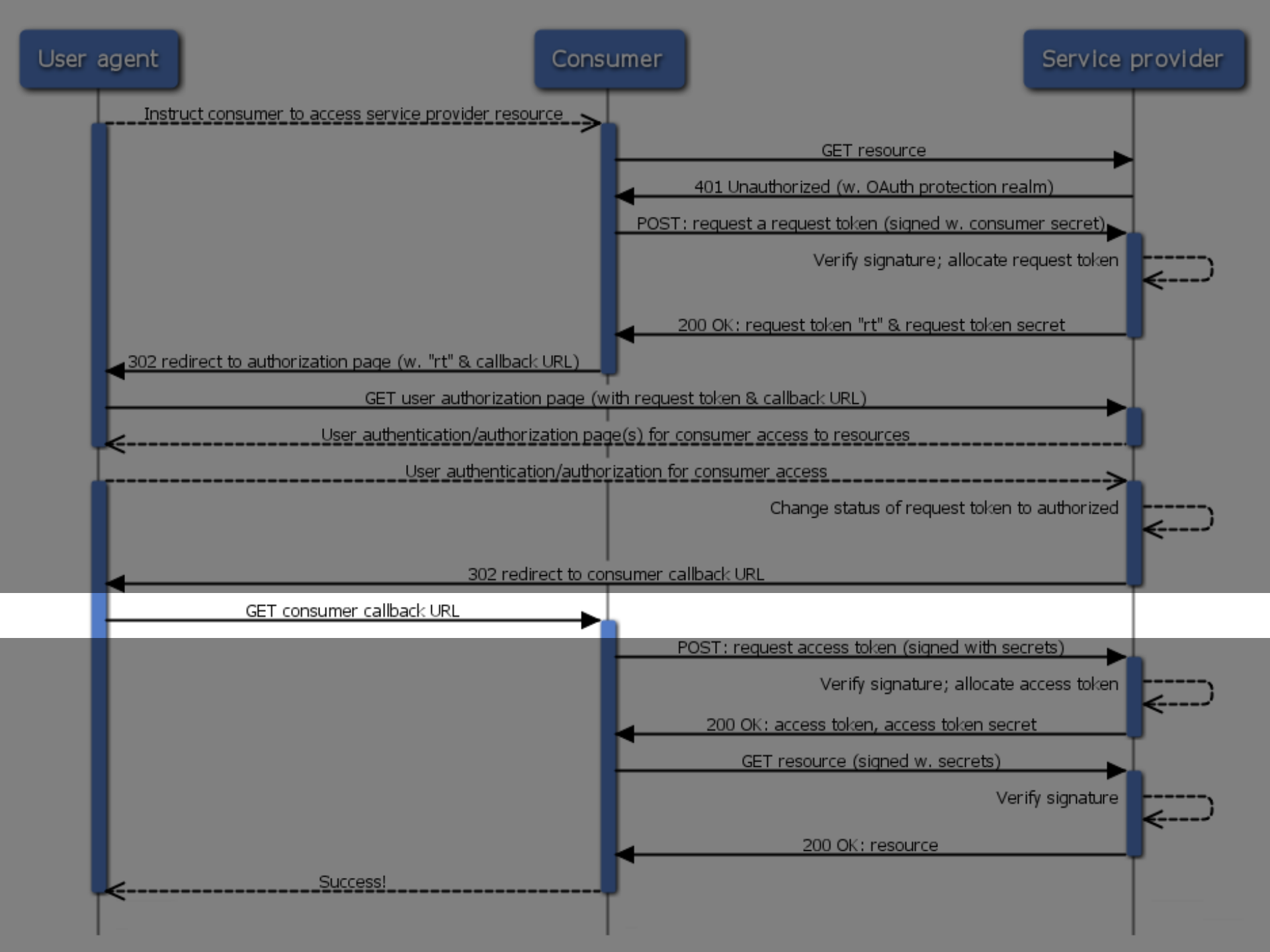

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

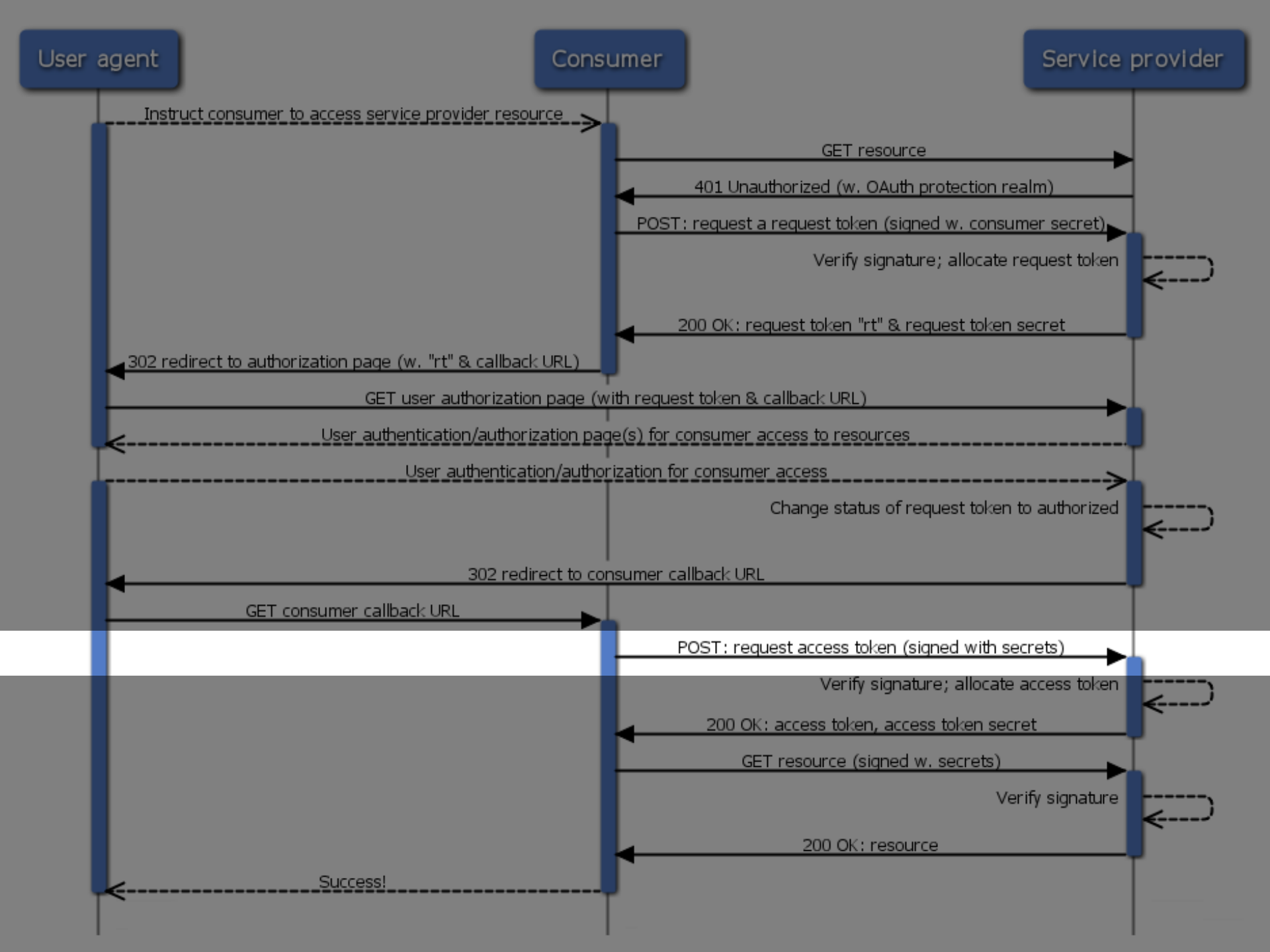

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

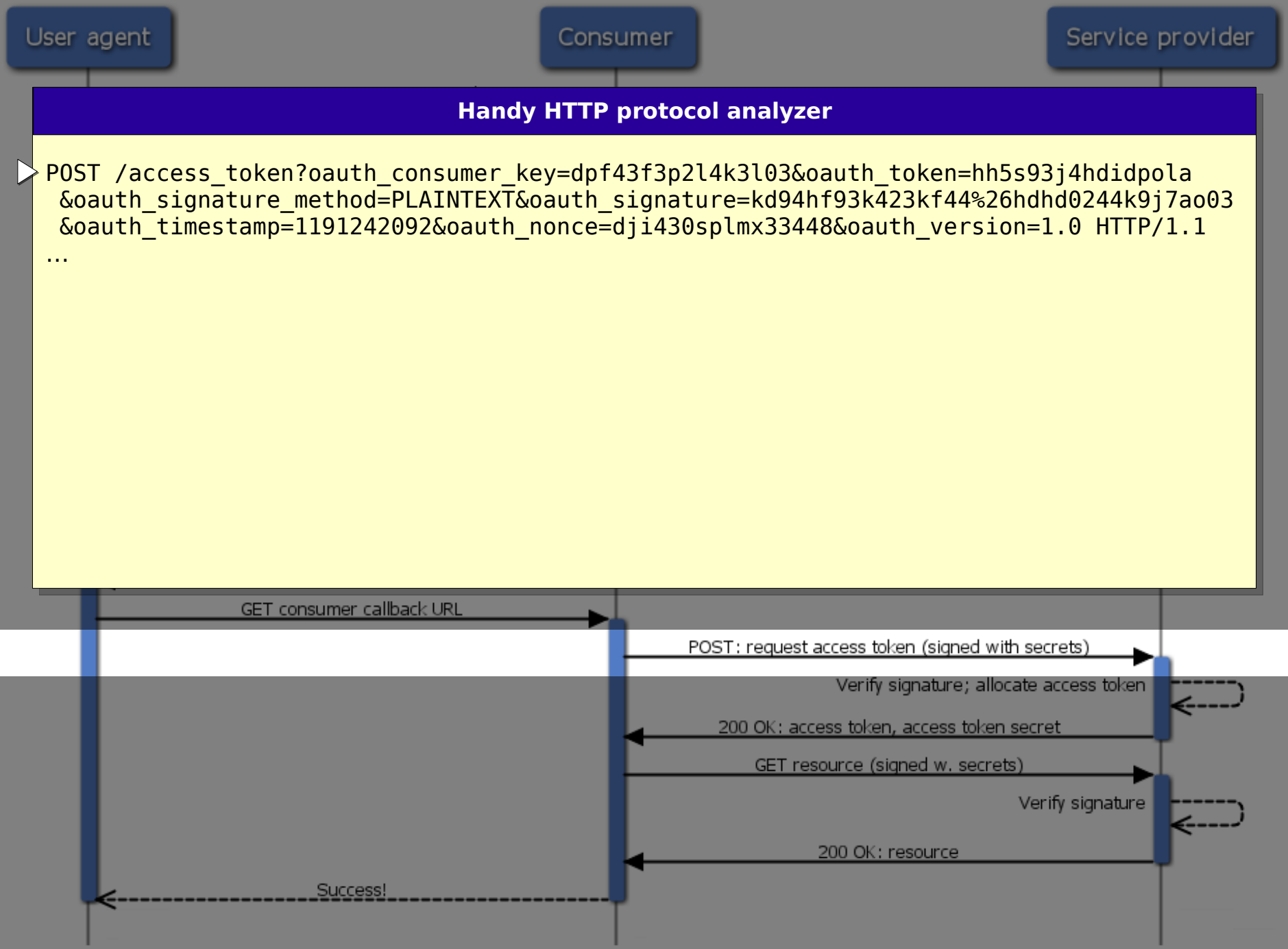

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

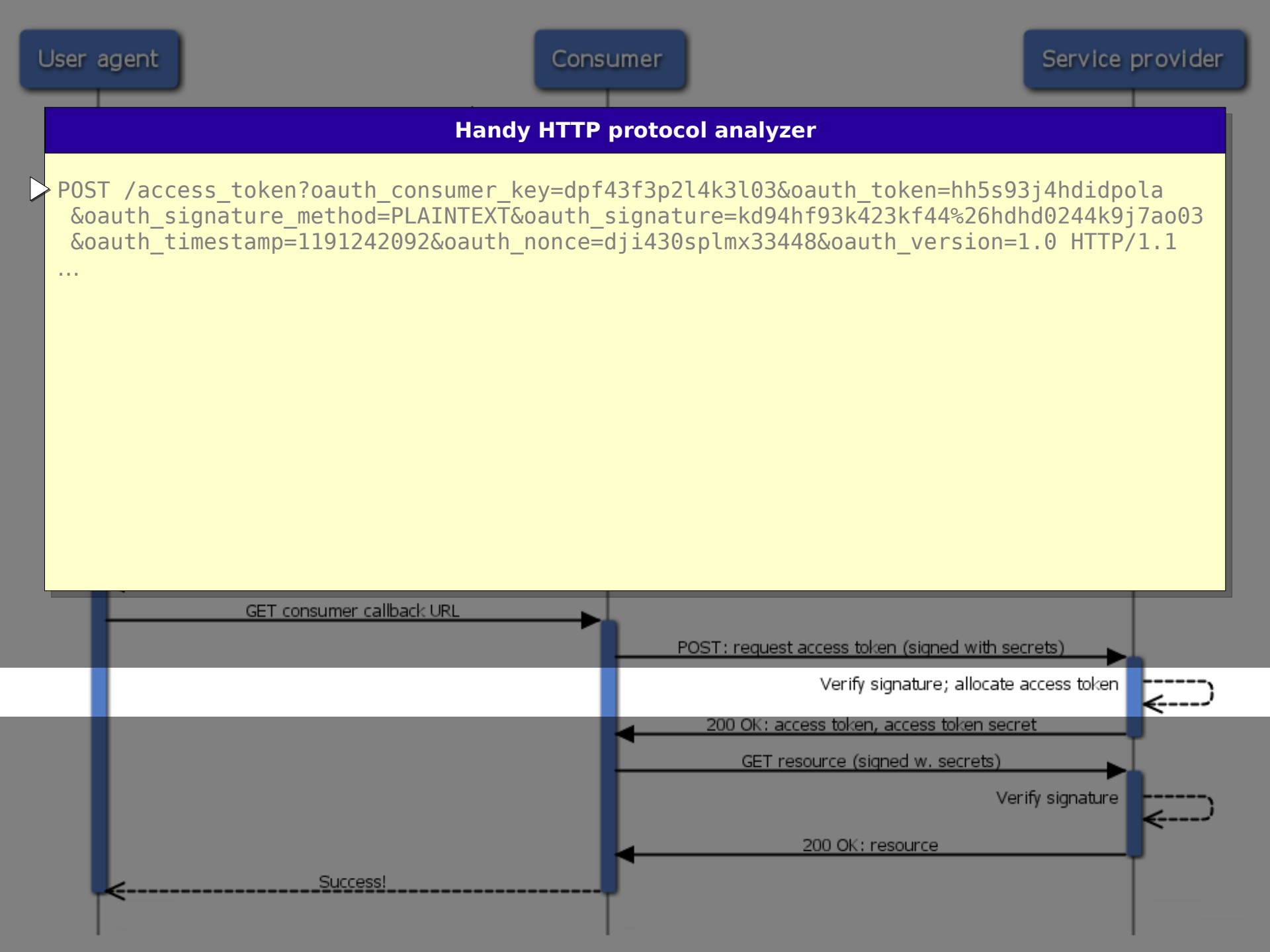

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

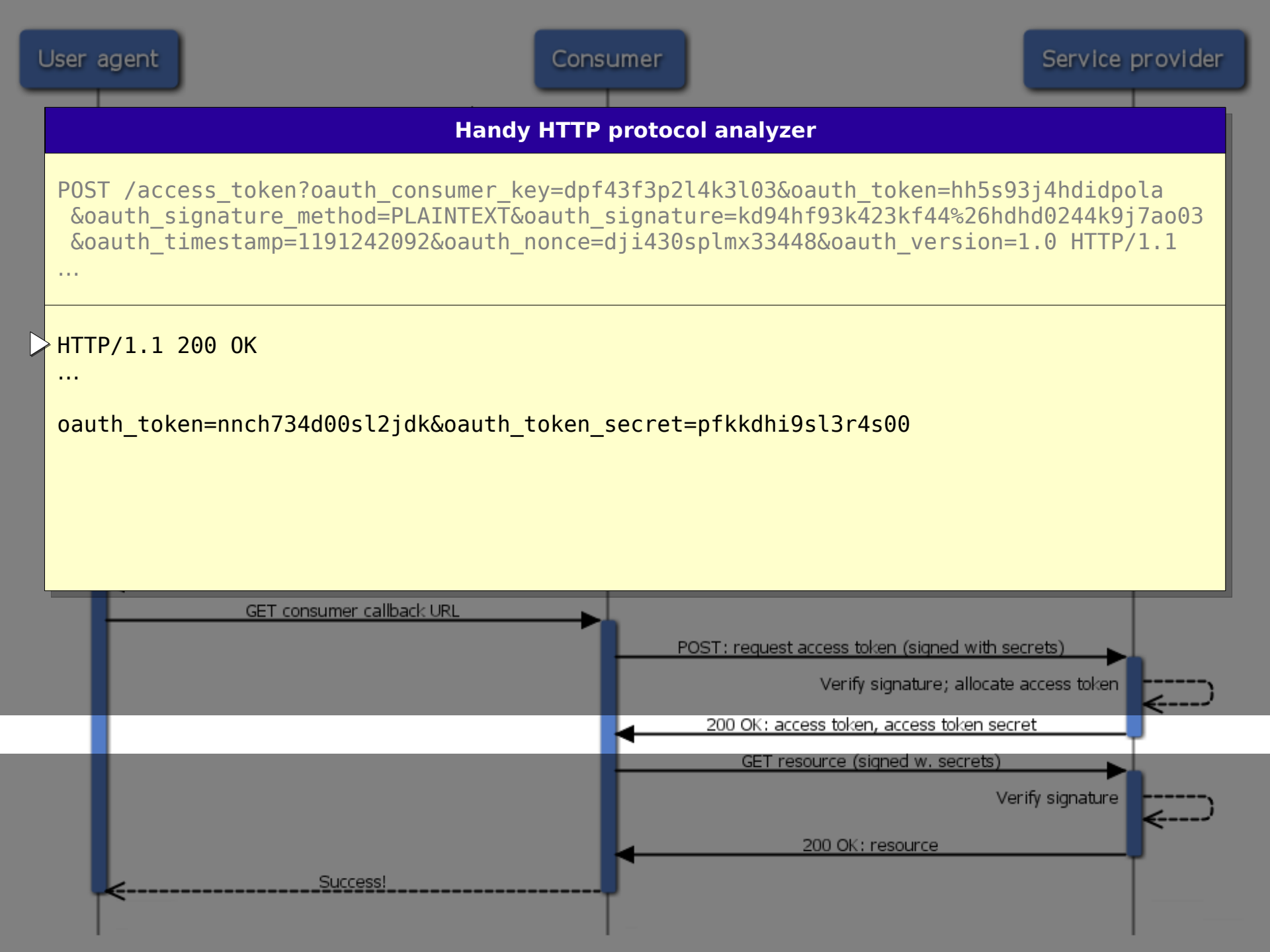

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

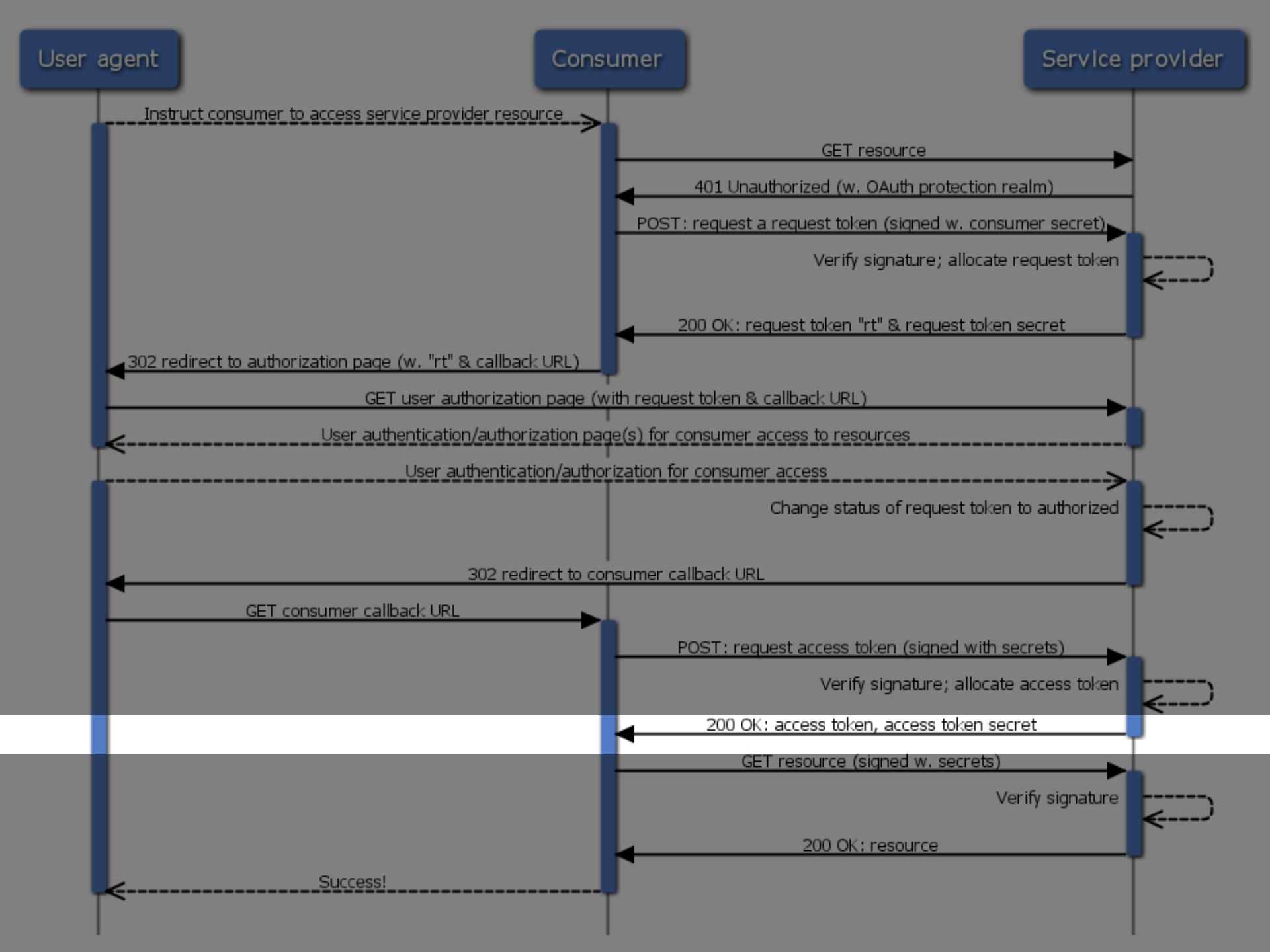

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

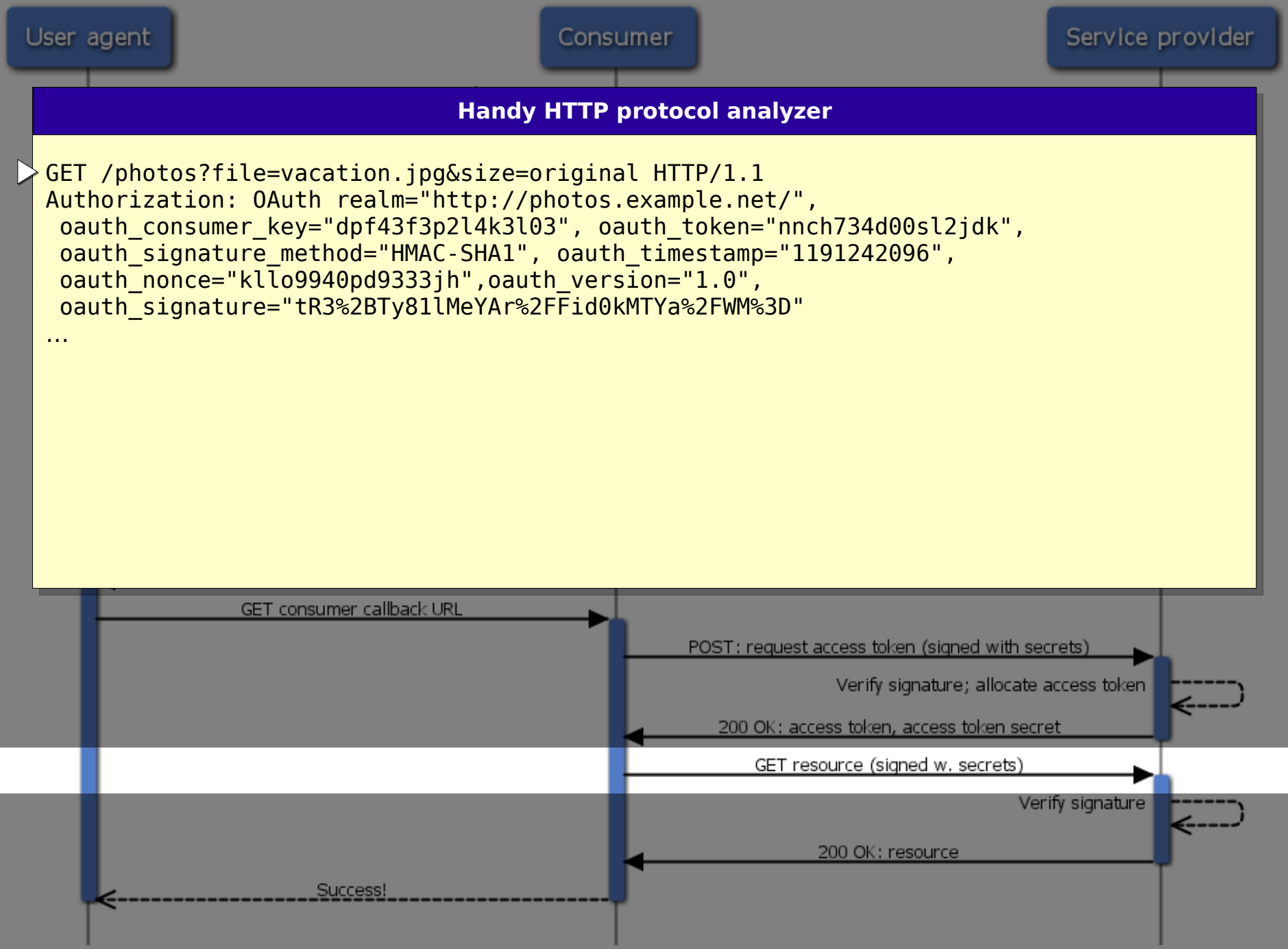

User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

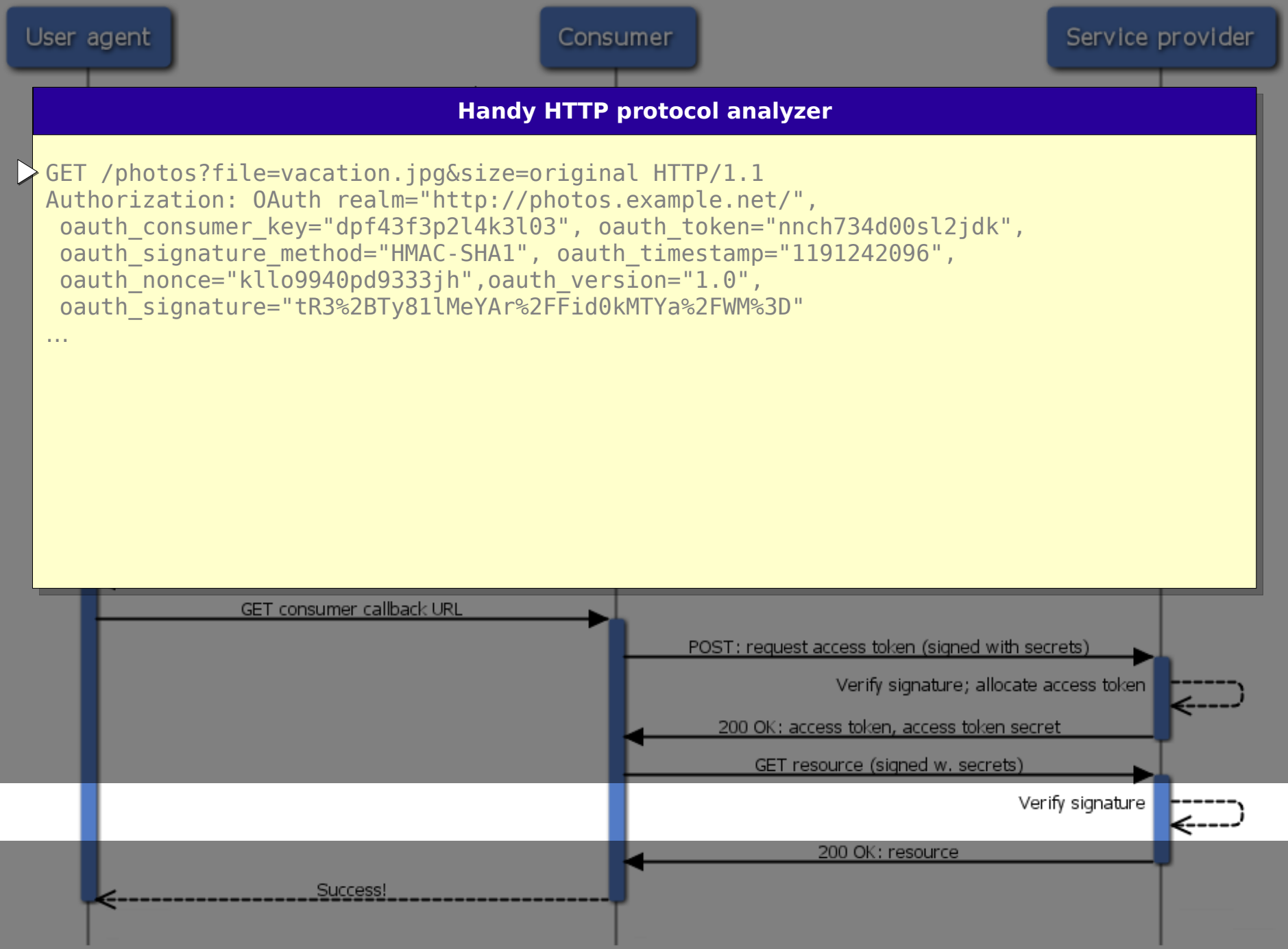

User agent
Consumer
Service provider
Handy HTTP protocol analyzer
POST /access_token?oauth_consumer_key=dpf43f3p2l4k3l03&oauth_token=hh5s93j4hdidpola
&oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26hdhd0244k9j7ao03
&oauth_timestamp=1191242092&oauth_nonce=dji430splmx33448&oauth_version=1.0 HTTP/1.1
...
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

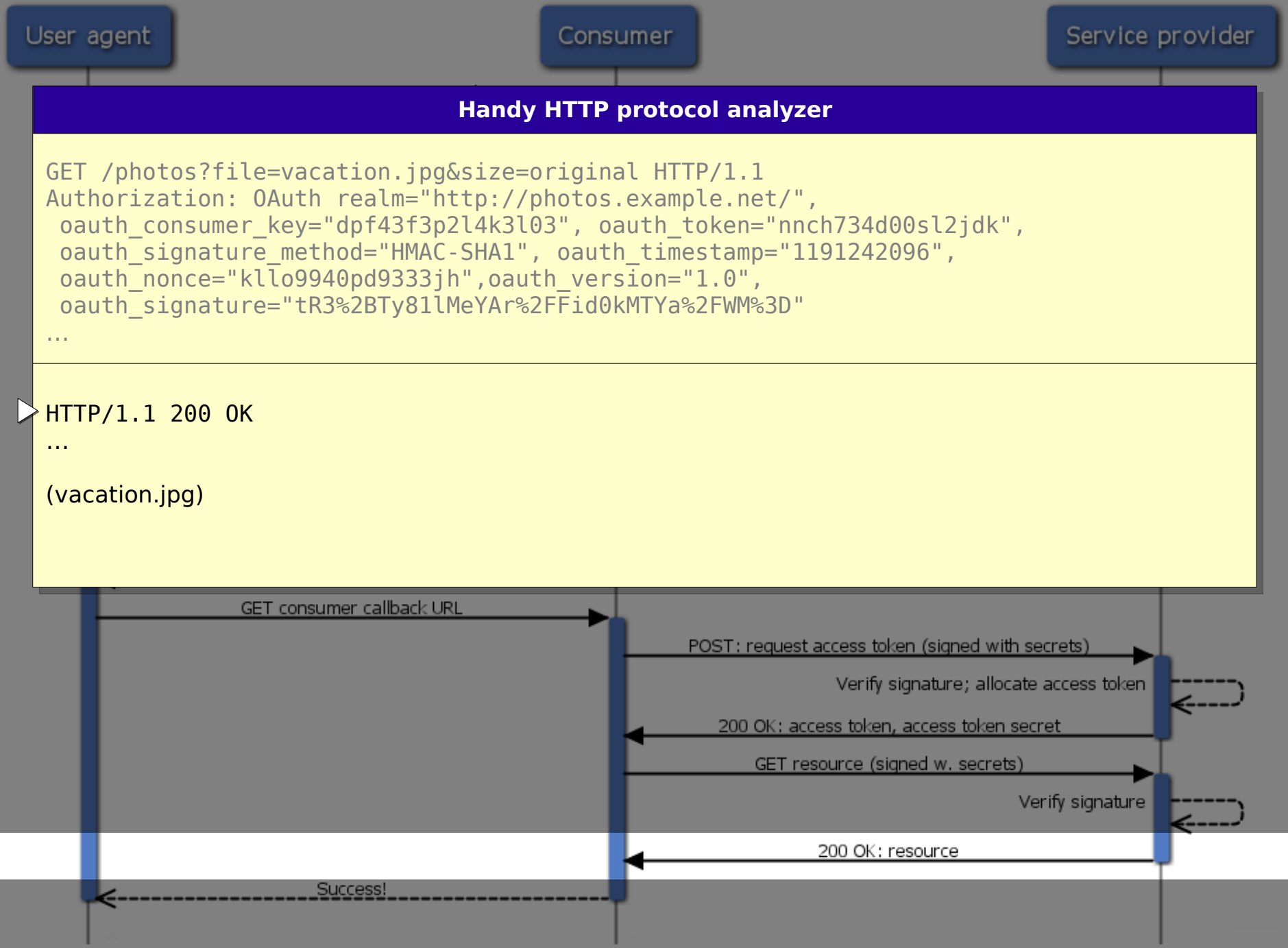

User agent
Consumer
Service provider
Handy HTTP protocol analyzer
POST /access_token?oauth_consumer_key=dpf43f3p2l4k3l03&oauth_token=hh5s93j4hdidpola
&oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26hdhd0244k9j7ao03
&oauth_timestamp=1191242092&oauth_nonce=dji430splmx33448&oauth_version=1.0 HTTP/1.1
...
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

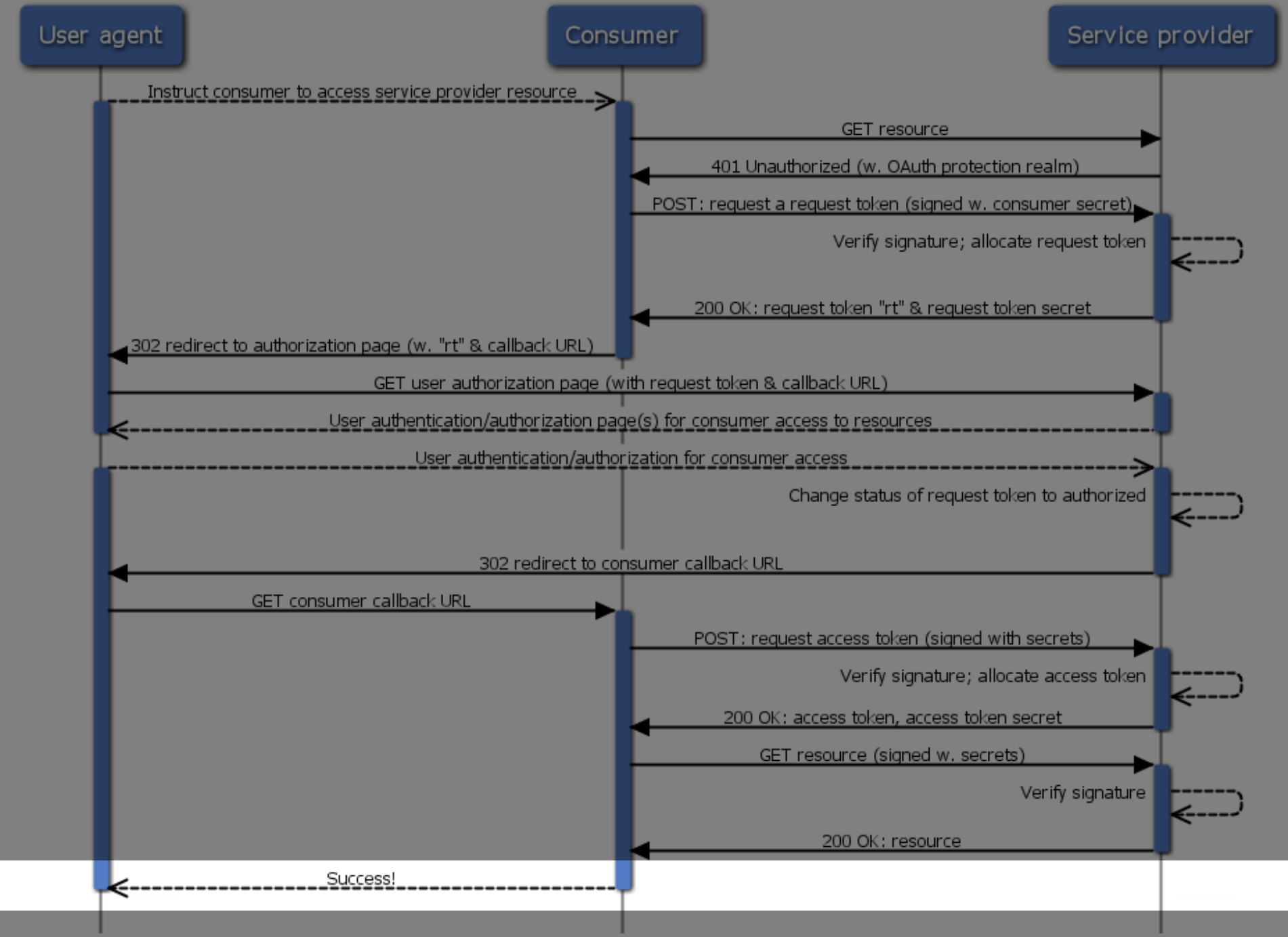

User agent
Consumer
Service provider
Handy HTTP protocol analyzer
POST /access_token?oauth_consumer_key=dpf43f3p2l4k3l03&oauth_token=hh5s93j4hdidpola
&oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26hdhd0244k9j7ao03
&oauth_timestamp=1191242092&oauth_nonce=dji430splmx33448&oauth_version=1.0 HTTP/1.1
...
HTTP/1.1 200 OK
...
oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Handy HTTP protocol analyzer
GET /photos?file=vacation.jpg&size=original HTTP/1.1
Authorization: OAuth realm="http://photos.example.net/",
oauth_consumer_key="dpf43f3p2l4k3l03", oauth_token="nnch734d00sl2jdk",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1191242096",
oauth_nonce="kllo9940pd9333jh",oauth_version="1.0",
oauth_signature="tR3%2BTy81lMeYAr%2FFid0kMTYa%2FWM%3D"
...
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Handy HTTP protocol analyzer
GET /photos?file=vacation.jpg&size=original HTTP/1.1
Authorization: OAuth realm="http://photos.example.net/",
oauth_consumer_key="dpf43f3p2l4k3l03", oauth_token="nnch734d00sl2jdk",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1191242096",
oauth_nonce="kllo9940pd9333jh",oauth_version="1.0",
oauth_signature="tR3%2BTy81lMeYAr%2FFid0kMTYa%2FWM%3D"
...
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Handy HTTP protocol analyzer
GET /photos?file=vacation.jpg&size=original HTTP/1.1
Authorization: OAuth realm="http://photos.example.net/",
oauth_consumer_key="dpf43f3p2l4k3l03", oauth_token="nnch734d00sl2jdk",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1191242096",
oauth_nonce="kllo9940pd9333jh",oauth_version="1.0",
oauth_signature="tR3%2BTy81lMeYAr%2FFid0kMTYa%2FWM%3D"
...
HTTP/1.1 200 OK
...
(vacation.jpg)
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!


User agent
Consumer
Service provider
Instruct consumer to access service provider resource
GET resource
401 Unauthorized (w. OAuth protection realm)
POST: request a request token (signed w. consumer secret)
Verify signature; allocate request token
200 OK: request token "rt" & request token secret
302 redirect to authorization page (w. "rt" & callback URL)
GET user authorization page (with request token & callback URL)
User authentication/authorization page(s) for consumer access to resources
User authentication/authorization for consumer access
Change status of request token to authorized
302 redirect to consumer callback URL
GET consumer callback URL
POST: request access token (signed with secrets)
Verify signature; allocate access token
200 OK: access token, access token secret
GET resource (signed w. secrets)
Verify signature
200 OK: resource
Success!

Why consider OAuth over others?

Mashups quickly evolve toward delegation model

Aligns very well with REST (use of HTTP header)

More secure than storing credentials everywhere

Flexible access token management capability

Already multiple client and server implementations

Strong community — now an IETF working group

Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

OpenSSO simple REST services APIs

OAuth overview

Protecting and consuming services with OAuth

Final thoughts

REST services with OAuth in Java

OAuth consumer components for Java:

- Jersey OAuth client filter

OAuth server components for Java:

- Jersey OAuth signature library
- Jersey OAuth server request object
- OpenSSO OAuth token service & servlet filter

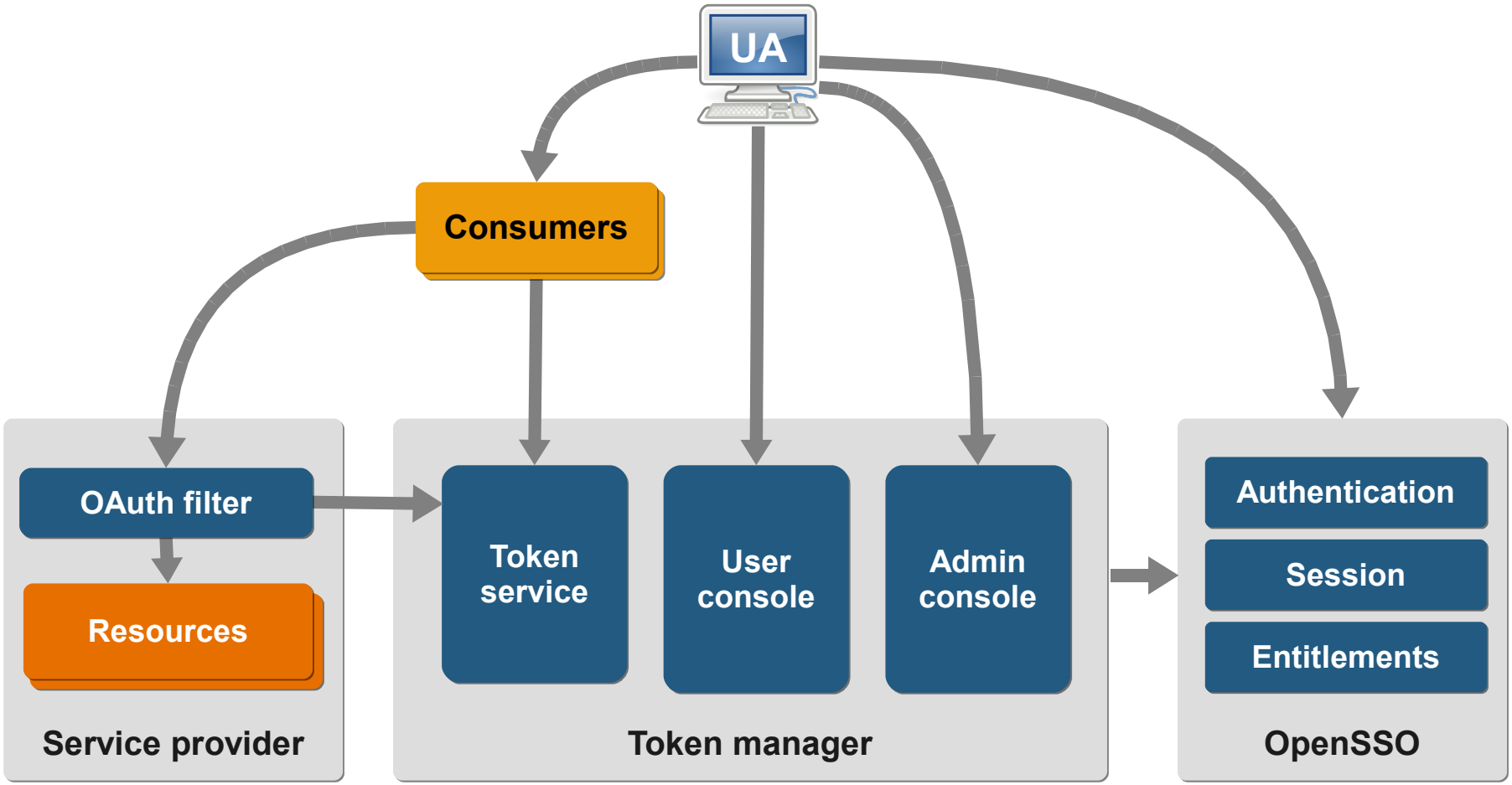
Jersey OAuth client filter

```
Client client = Client.create();
WebResource resource = client.resource("http://example.com/base");

...

OAuthSecrets secrets = new OAuthSecrets().consumerSecret("...").
 tokenSecret("...");

OAuthParameters params = new OAuthParameters().consumerKey("...").
 token("...").signatureMethod("HMAC-SHA1");

resource.addFilter(new OAuthClientFilter(
 client.getProviders(), params, secrets));

...

response = resource.get(...);
```

Jersey OAuth server request object

```
@GET public String handle(@Context HttpContext hc) {

    OAuthServerRequest osr = new OAuthServerRequest(hc.getRequest());
    OAuthParameters params = new OAuthParameters().readRequest(osr);
    OAuthSecrets secrets = new OAuthSecrets();

    ...
    (populate secrets for consumer key and access token)
    ...

    boolean verified = OAuthSignature.verify(osr, params, secrets);

     ...
}
```

OpenSSO: OAuth architecture overview



Agenda

What is OpenSSO

Problem description

REST overview

REST security overview

OpenSSO simple REST services APIs

OAuth overview

Protecting and consuming services with OAuth

Final thoughts

Final thoughts

- Use filter pattern for security
  - Keep security code separate from application code
- Use tokens instead of username and password for service access
- Be careful with your tokens, especially secrets
  - Use HTTPS to transfer sensitive information
- Leverage existing specifications and protocols and implementations
- Avoid breaking rest, avoid passing thru query parameters... prefer headers

Final thoughts

- Become familiar with message-level security and digital signatures
  - Java libraries make it easy
- Determine requirements for different resources
  - Distinguish user data from general data
  - Leverage delegation models
- Get familiar with OAuth

More OpenSSO at JavaOne

TS-4012

Pragmatic identity 2.0: simple, open, identity services using REST

Thursday @ 10:50 AM

LAB-6727

Web application security with OpenSSO: from simple login to single sign on to federation

Thursday @ 1:30 PM

BOF-4903

A RESTful approach to identity-based web services

Thursday @ 7:30 PM

Resources on the Intertube

OpenSSO project

www.opensso.org

Jersey project

jersey.dev.java.net

OAuth protocol

oauth.net

JavaOne℠

Thank You

Paul C. Bryan
Sean Brydon
Aravindan Ranganathan

pbryan@sun.com
Sean.Brydon@sun.com
aravind@sun.com