
Securing Microservices

Good Ole' HTTP Authentication

- HTTP Basic and Digest Authentication: IETF RFC 2617
- Have been around since 1996 (Basic)/1997 (Digest)
- Pros:
  - Respects Web architecture:
    - stateless design (retransmit credentials)
    - headers and status codes are well understood
  - Does not prohibit caching (set `Cache-Control` to `public`)
- Cons:
  - Basic Auth must be used with SSL/TLS (plaintext password)
  - Not ideal for the human Web – no standard logout
  - Only one-way authentication (client to server)

HTTP Basic Auth Example

1. Initial HTTP request to protected resource

```
GET /index.html HTTP/1.1
Host: example.org
```

2. Server responds with

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="MyRealm"
```

3. Client resubmits request

```
GET /index.html HTTP/1.1
Host: example.org
Authorization: Basic Qm9iCnBhc3N3b3JkCg==
```

Further requests with same or deeper path can include the additional `Authorization` header preemptively

HTTP Digest Difference

- Server reply to first client request:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
      realm=myrealm@example.org,
      qop="auth,auth-int",
      nonce="a97d8b710244df0e8b11d0f600bfb0cdd2",
      opaque="8477c69c403ebaf9f0171e9517f347f2"
```

- Client response to authentication challenge:

```
Authorization: Digest
      username="bob",
      realm=myrealm@example.org,
      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
      uri="/index.html",
      qop=auth, nc=00000001, cnonce="0a6f188f",
      response="56bc2ae49393a6589745097850 7ff442",
      opaque="8477c69c403ebaf9f0171e9517f347f2"
```

WSSE Authentication

- Driven from the Atom community
- Use the WS-Security Username Token profile mapped to HTTP headers
- Doesn’t pass sensitive data in clear text
- Does require both sides to know a shared secret

Man-in-the-Middle

- All HTTP Authentication schemes can be hijacked by a man-in-the-middle attack
- Can intercept a Digest response from a service and change it into a Basic challenge
- Basic is easy to crack, attacker learns the password

Transport level security considered mandatory when you're using HTTP authentication of any variety

SSL / TLS

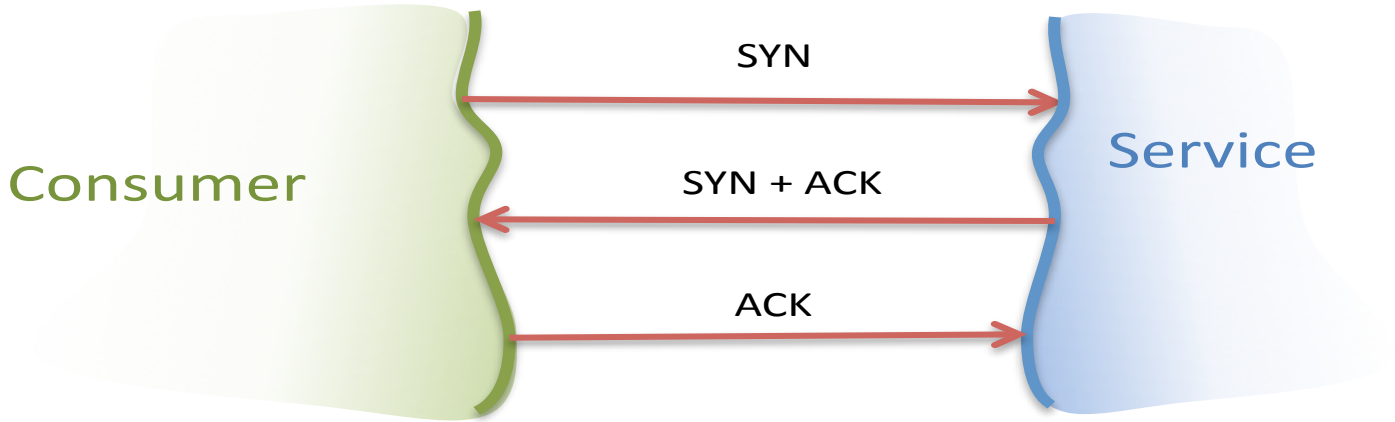
- “Strong” server and optional client authentication, confidentiality and integrity protection
- The only feasible way to secure against man-in-the-middle attacks
- Not broken! Even if some people like to claim otherwise
- Not very cache friendly though…

Transport Level Security

- TLS is the successor to Netscape's original SSL
- Tightened up some security loopholes
- Now under IETF's stewardship
  - RFC 2246
  - RFC 5246
- Provides a secure channel between client and server
  - Authenticated
  - Identified (bilateral too)
  - Confidential
  - Integrity assured

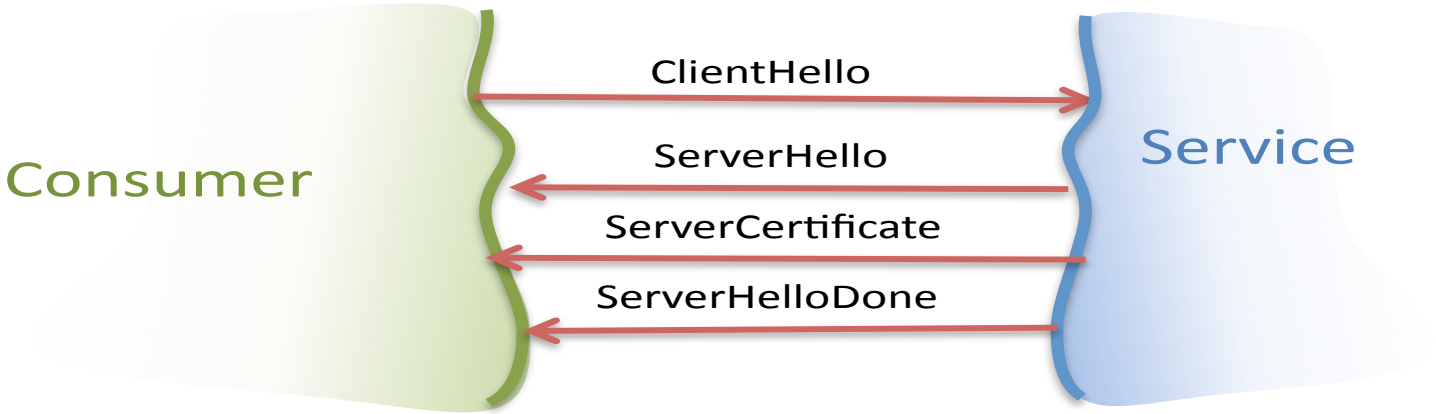
Start with a TCP Handshake

- Classic handshake the underpins HTTP connections
- Not secure at this point, obviously



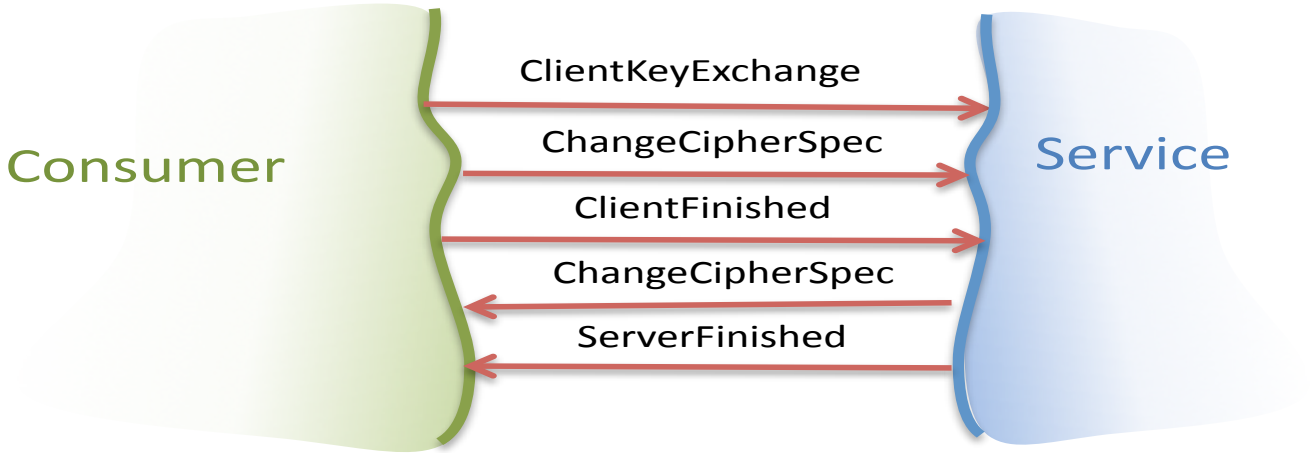
Negotiate some cryptographic options

- Client sends its hashing/encryption capabilities and preferences to the server
- Server responds with its choices from those the client presented
  - Preferring highest levels of security
- Server sends its certificate
  - Public key, CA
- Server indicates that it's complete



Switch on the crypto

- Client uses server's public key to send a PreMasterKey to the server
  - If the server is authenticated, it can decrypt this secret with its private key
- Client ChangeCipherSpec flips client onto the secure channel
- ClientFinished message sends a hash of the entire conversation
  - To alleviate any possibility of missing/tampered messages
- Server ChangeCipherSpec flips server onto the secure channel
- ServerFinished sends a hash of the entire conversation to the client



Network and performance considerations

- Just use HTTPs everywhere?
- Reduces options for caching
  - Reverse proxies and client-side caching only
- Expensive to set up connections
  - Though relatively cheap to maintain, if you have enough sockets
- Securing a channel on the risk/value profile of a resource
  - Secure channels only for high value/risk resources
- Can use a hybrid approach…

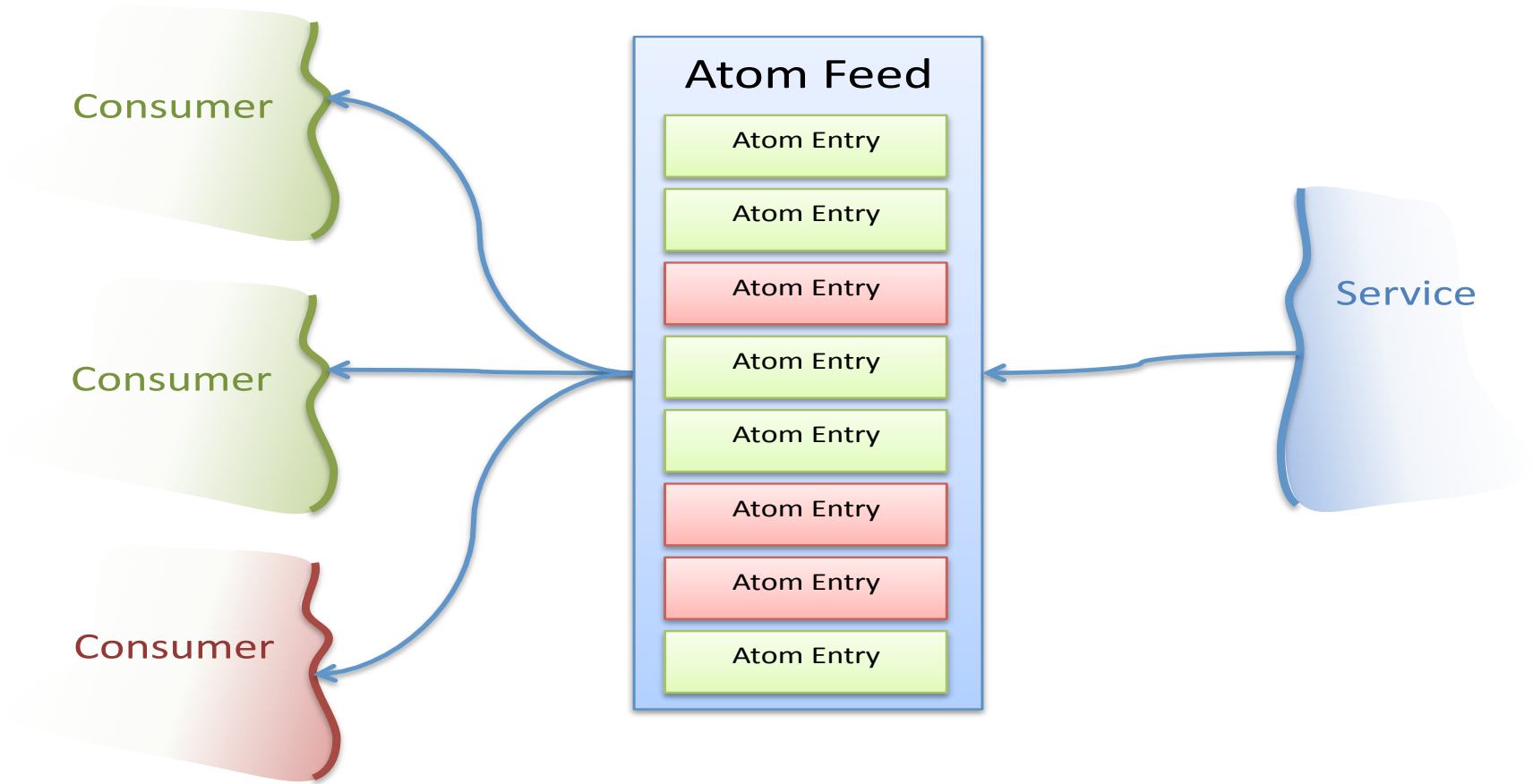
Publish secret data

- Here are my bank account details, please feel free to use them:

```
bb0ff34c3ab9c2a478cb7b8b61139a787bab5de1b4b
5ec463db070e1b72c502114758f1afd44c09b799207
3ccf00b43dc991579ddc5cbb91ea6984cbda08be9f
```

- Useless to you, or anyone else, unless you know the decryption key

Widely publish secret data



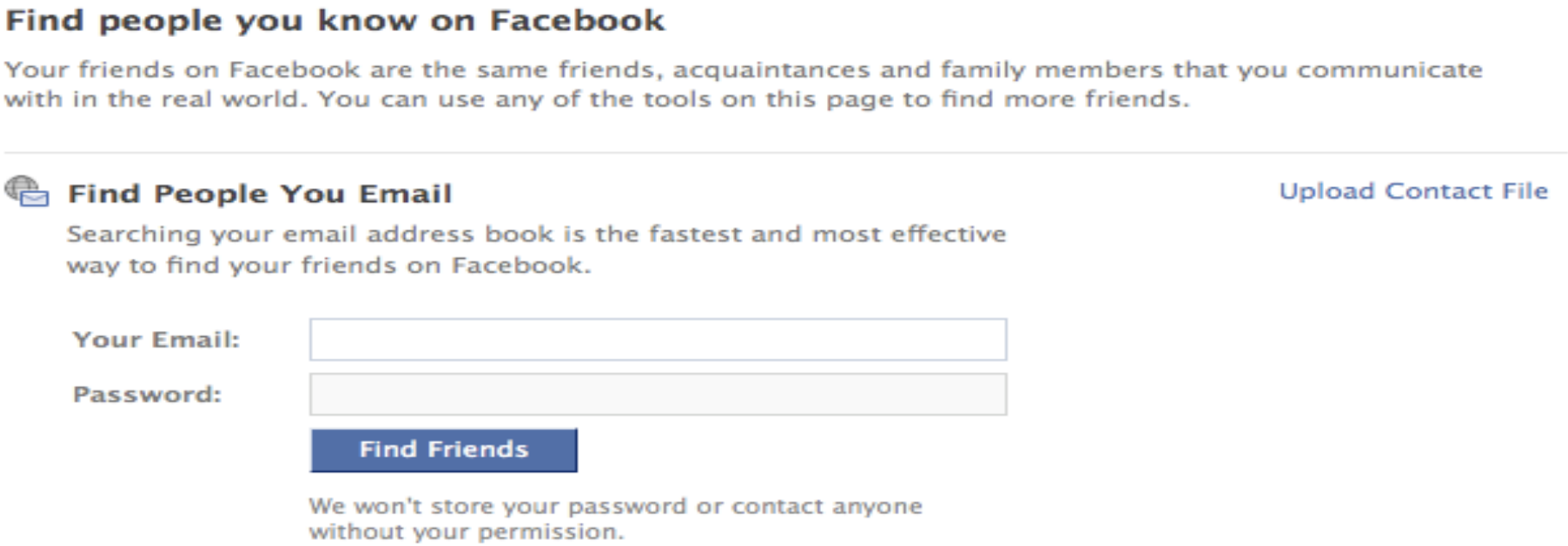
Secure messaging with Atom

- The contents of individual atom:entry elements can be encrypted with public/shared keys for specific consumers
- Only consumers who know the corresponding private/shared key can make sense of the content
  - To anyone else, it's gibberish
- Keep the crypto strong!
  - This will be in the public domain, beware brute force on weak algorithms
- Can cache this widely, reduced performance hit
- But beware coupling via keys!

Beware cheap rental GPUs that make brute-force attacks feasible for a small rental cost


Authorisation and OAuth

Why OAuth?


Find people you know on Facebook
Your friends on Facebook are the same friends, acquaintances and family members that you communicate with in the real world. You can use any of the tools on this page to find more friends.
Find People You Email
Upload Contact File
Searching your email address book is the fastest and most effective way to find your friends on Facebook.
Your Email:
Password:
Find Friends
We won't store your password or contact anyone without your permission.

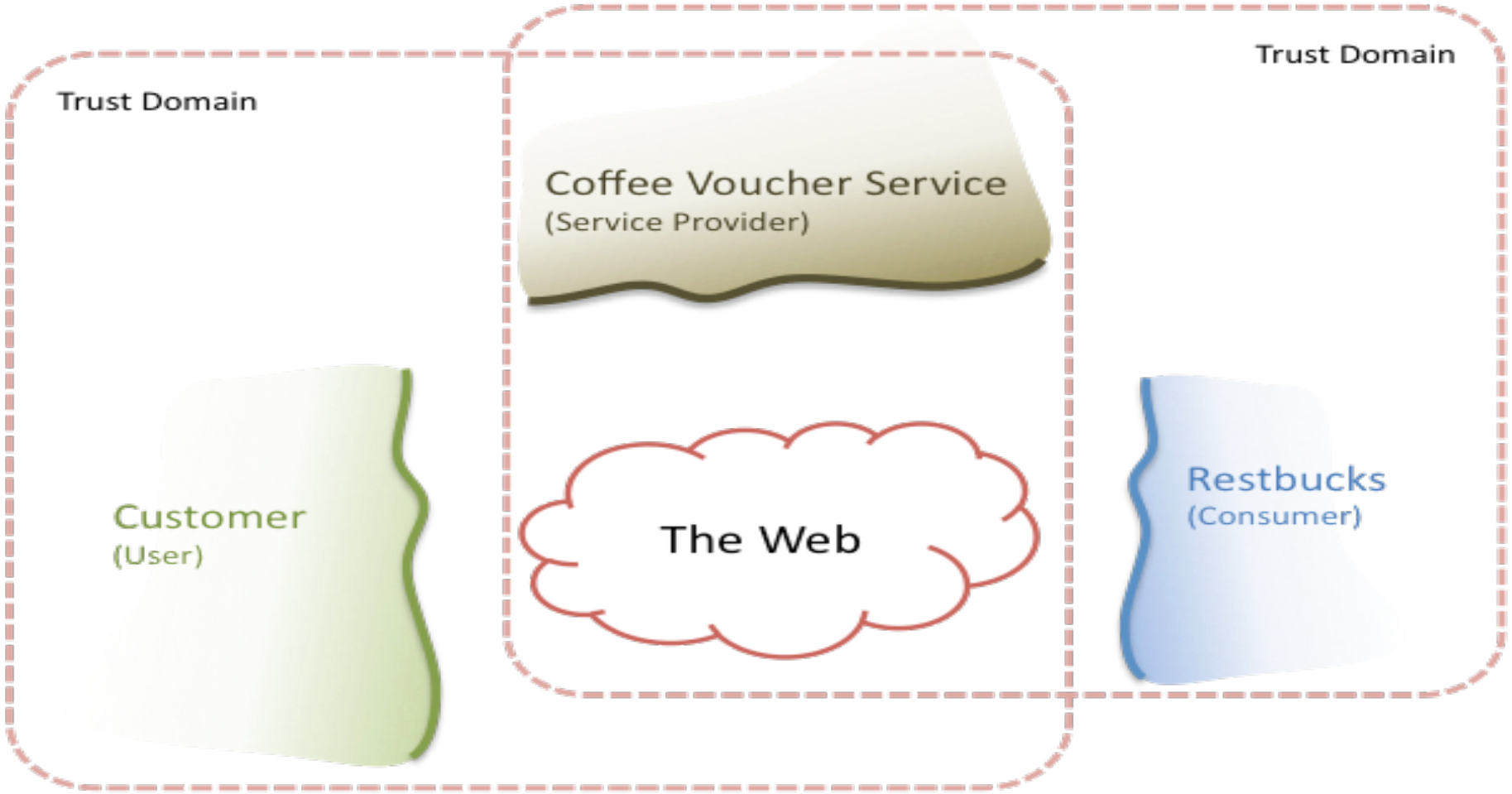
Restbucks Electronic Wallet

Business problem:
Customers ask for credit and are routinely refused (and possibly assaulted) while our legal costs are mounting

Solution:
Provide an electronic prepay mechanism

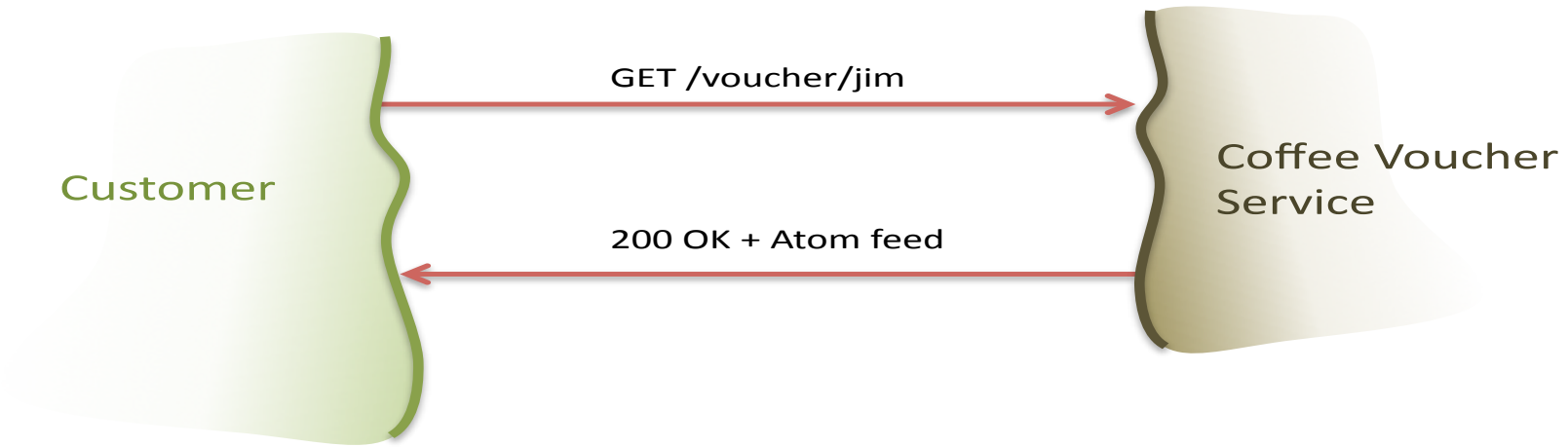
Constraints:
We can't afford to implement the prepay scheme, we need to partner

OAuth Establishes Trust Domains



Check your wallet

- Check which vouchers you have left to spend
- Use Atom for lists of vouchers…



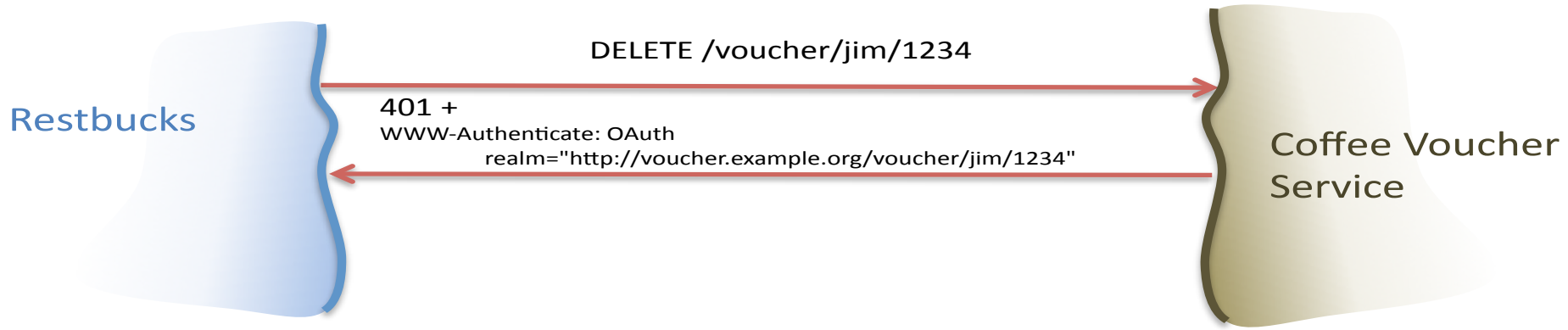
Start the payment

```
PUT /payment/1234 HTTP/1.1
Accept: application/vnd.restbucks+xml
Content-Type: application/vnd.restbucks+xml
User-Agent: Java/1.6.0_17
Host: restbucks.com
Connection: keep-alive
Content-Length: 205

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<voucherPayment xmlns="http://schemas.restbucks.com">

   <voucherUri>http://vouchers.example.org/voucher/jim/
   1234</voucherUri>
</voucherPayment>
```

Restbucks tries to redeem the voucher, gets refused



On the wire…

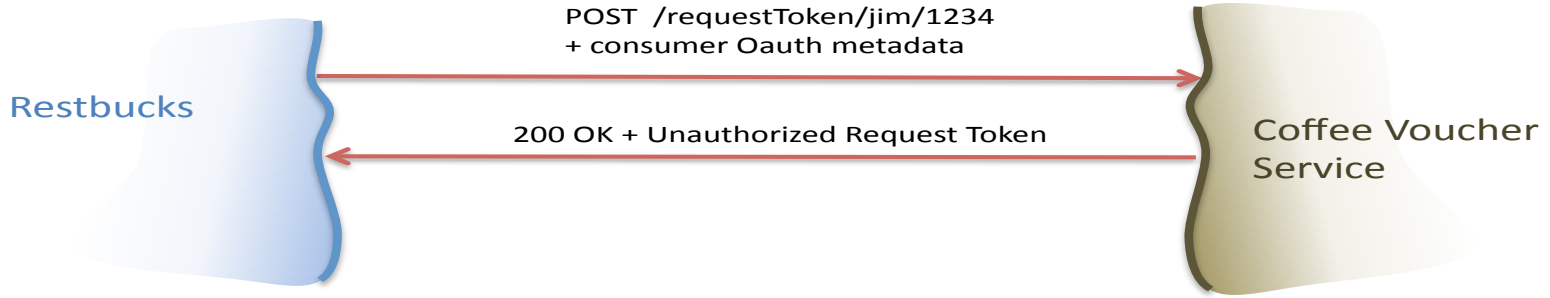
- Restbucks attempts to redeem the voucher

```
DELETE /voucher/jim/1234 HTTP/1.1
Accept:
   application/vnd.coffeevoucher+xml
Host: vouchers.example.org
Connection: keep-alive
```

- Voucher service challenges the redemption

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: OAuth
   realm="http://vouchers.example.or
   g/"
Content-Type: application/x-www-
   form-urlencoded
Date: Sat, 03 Apr 2010 00:27:47 GMT
```

Restbucks demands a *request token*




```
POST /requestToken/voucher/jim/1234 HTTP/1.1
Accept: application/x-www-form-urlencoded
Authorization: OAuth oauth_callback="http%3A%2F%2Frestbucks.com%2Fpayment%2F9baea738",
    oauth_signature="GHU4a%2Fv9JnvZFTXnRiVf3HqDGfk%3D", oauth_version="1.0",
    oauth_nonce="05565e78", oauth_signature_method="HMAC-SHA1", oauth_consumer_key="light",
    oauth_timestamp="1270254467"
Host: vouchers.example.org
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
Content-Length: 79
Date: Sat, 03 Apr 2010 00:27:47 GMT

oauth_token=b0c2ec2c&oauth_token_secret=f41eab9d&oauth_callback_confirmed=true
```

Customer is redirected by Restbucks to Voucher Service

```
HTTP/1.1 303 See Other
Location:
  http://vouchers.example.org/signIn/voucher/jim/1234?oauth_token=b0c2ec2c
Content-Type: application/vnd.restbucks+xml
Content-Length: 0
Date: Sat, 03 Apr 2010 00:27:47 GMT
```

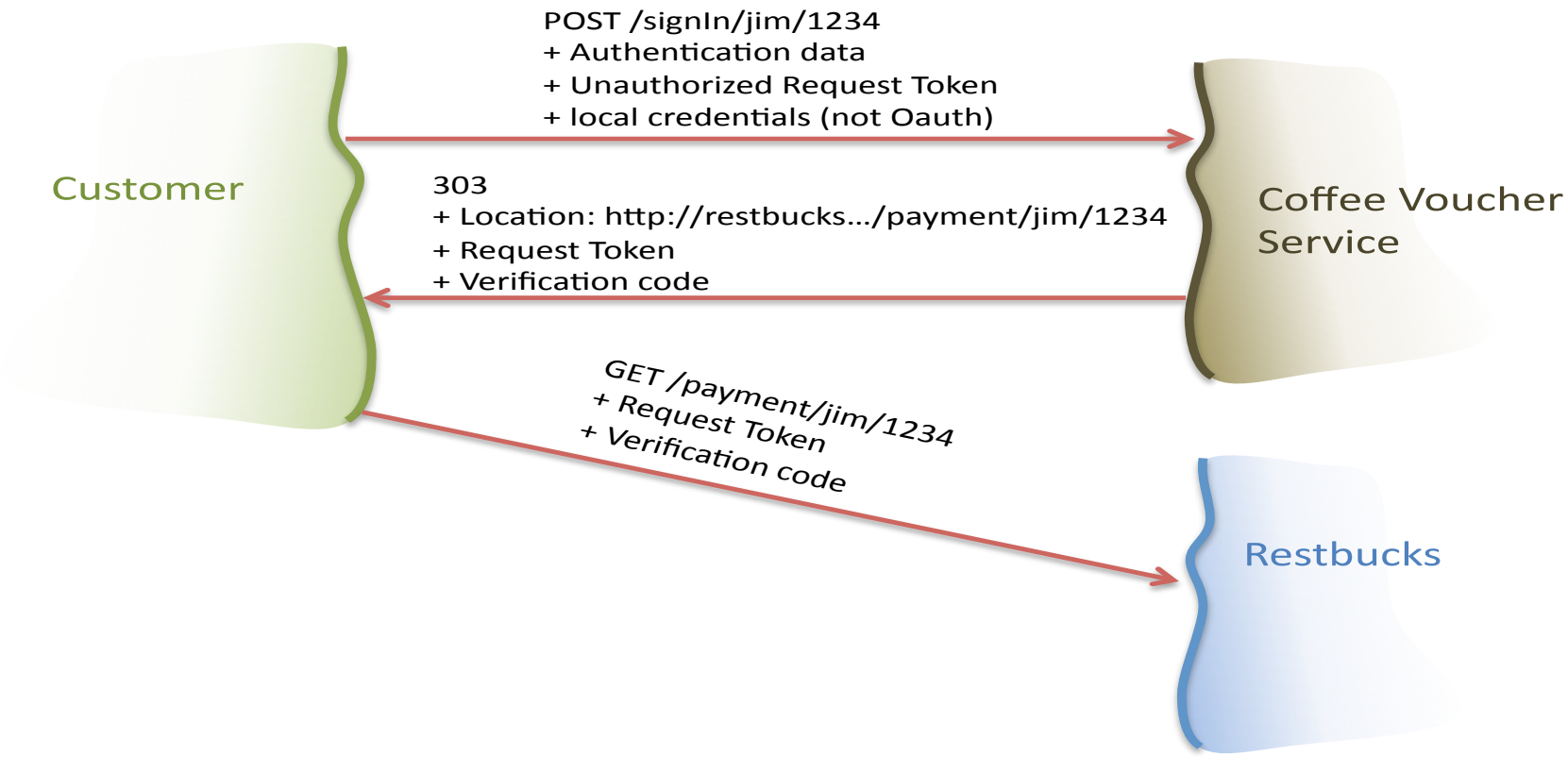
Customer signs in and authorises a voucher

```
GET /signIn/voucher/jim/1234?oauth_token=b0c2ec2c HTTP/1.1
Accept: application/xhtml+xml
Host: vouchers.example.org
Connection: keep-alive
```


```
HTTP/1.1 200 OK
Content-Type: application/xhtml+xml
Content-Length: 360
Date: Sat, 03 Apr 2010 00:27:47 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <body>
  <form action="http://vouchers.example.org/signIn/voucher/jim/1234"
    method="post">
    <input type="hidden" name="oauth_token" value="b0c2ec2c"/>
    <input type="password" name="password"/>
  </form>
  </body>
</html>
```

Successful sign in and redirect to Restbucks



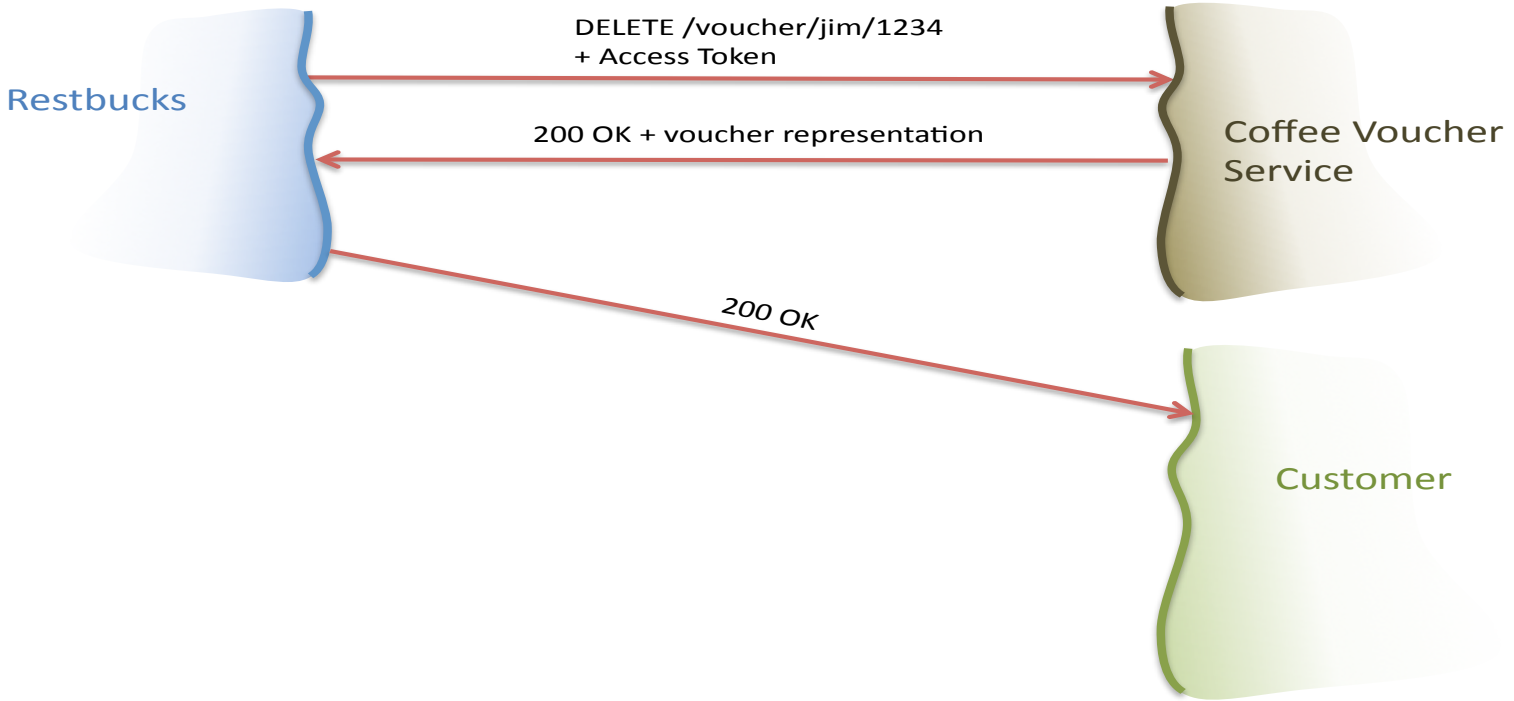
Restbucks demands an *Access token*


```
POST /accessToken/voucher/jim/1234 HTTP/1.1
Content-Type: application/x-www-form-
    urlencoded
Authorization: OAuth
    oauth_signature="m7ials2v0VJuKD05BrNGISi7
    Nog%3D", oauth_version="1.0",
    oauth_nonce="10d13b8e",
    oauth_signature_method="HMAC-SHA1",
    oauth_consumer_key="light",
    oauth_verifier="c87677a4",
    oauth_token="b0c2ec2c",
    oauth_timestamp="1270254468"
Host: vouchers.example.org
Accept: application/x-www-form-urlencoded
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Content-Type:
  application/x-www-form-
  urlencoded
Content-Length: 49
Date: Sat, 03 Apr 2010
  00:27:48 GMT

oauth_token=99fe97e1&oaut
  h_token_secret=255ae587
```

Restbucks uses Access token to redeem voucher



OAuth ends

```
DELETE /voucher/jim/1234 HTTP/1.1
Accept: application/vnd.coffeevoucher+xml
Authorization: OAuth
   oauth_signature="k2awEpciJkd2X8rt3NmgDg8AyUo%3D",
   oauth_version="1.0", oauth_nonce="9ceea445",
   oauth_signature_method="HMAC-SHA1",
   oauth_consumer_key="light", oauth_token="99fe97e1",
   oauth_timestamp="1270254468"
Host: vouchers.example.org
 HTTP/1.1 200 OK
 Content-Type: application/vnd.coffeevoucher+xml
 Content-Length: 252
 Date: Sat, 03 Apr 2010 00:27:48 GMT

 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 <link xmlns="http://voucher.example.org/schema"
    mediaType="application/vnd.coffeevoucher+xml"
    uri="http://vouchers.example.org/voucher/jim/1234" rel="voucher"/>
```

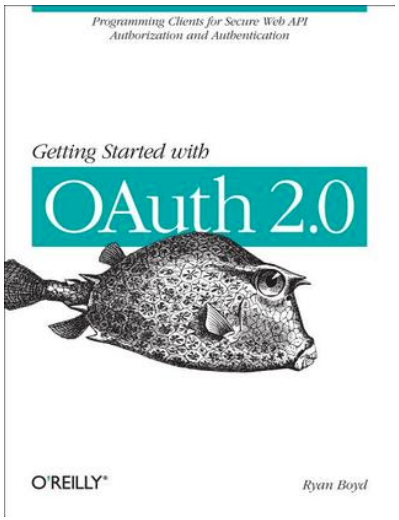
OAuth recommendations

- Useful for third party authorisation in private service interfaces
- **Invaluable** for third party authorisation in public services
- It's now standardised as RFC 5849
  - But there are numerous similarly named versions

- Can be tricky to implement
  - Particularly token management in the provider
  - Toolkits help a little, but crypto is never far from being **your** problem

OAuth 2.0

- Standard finalised October 2012
  - Plenty of standards politics around OAuth WRAP being divergent: Google, Microsoft, Salesforce.com and Yahoo!
- HTTPs mandatory
  - Good – crypto is no longer your problem, it's a transport issue. Less likely to inadvertently create security holes.
  - Bad – blanket crypto costs lots.
- Not universally liked, lead author has disassociated himself:
  - http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/
- Oauth 2.0 is now part of the toolkit for consumer web, it's likely we'll need it in the system-of-services soon



Tech Interlude

Service Hacks and Defences

Linkjacking

- We've used predictable links throughout this tutorial
  - /orders/1234
- They're good for learning, but they're easy to guess
- Malicious folks can guess your URIs to cause havoc (or worse)

- Using HTTPs is always an option
- Using unguessable URIs is fine for non-critical links
  - E.g. http://restbucks.com/b84fms9z

- Strike a reasonable balance between security and cost

Denial of Service

- Large incoming representations can cause problems
  - DoS through memory consumption on the server
- Use the `Content-Length` header strictly
  - No header, bin payload
    - 400 Bad Request
  - Stop processing payload after the number of bytes in the header
  - Bin payloads for suspiciously large headers
    - Who wants a million cappuccinos?
- Swallow `OutOfMemoryError`

```
POST /order HTTP/1.1
Host: restbucks.com
Content-Type:application/vnd.restbucks+xml

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>latte</name>
    <quantity>1</quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
<!-- Millions more item elements -->
  <item>
    <name>latte</name>
    <quantity>1</quantity>
    <milk>skim</milk>
    <size>small</size>
  </item>
</order>
```

Keep Secrets, Secret

- You can be a good guy, and help attackers
  - E.g. 401
  - Says there's something interesting here!
- Or you can be less helpful
  - 404
  - Says there's nothing to see here, even if there really is
- Don't use easily guessable URIs – they can be hijacked
  - UUID is your friend
- Think carefully about what attackers can learn from probing you
  - You don't always want to be a good guy!

Act Defensively

- Validate the content of representations
  - Just over 2^{16} café lattes would be quite lucrative
  - But is likely a ruse to get a large negative number into our workflow
    - Integer overflow?
- Don't forget anti-corruption layering between your resources and you domain model
  - REST is not mindlessly exposing a domain over HTTP!

```
<order
   xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>latte</name>
    <quantity>
      2147483648
    </quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
  ...
</order>
```

Don’t be gamed

```
GET /order/../../../../etc/passwd
HTTP/1.1
Host: restbucks.com
```

- Oh oh
- We just gave up the password file
- And rainbow tables cracked it in no time

```
GET /order/../../../../dev/random
HTTP/1.1
Host: restbucks.com
```

- Oh oh
- We just generated a never-ending stream of bytes
- And now we’re going to spend all our time serving them
- Also depleted the psuedorandom crypto pool
  - Can I even SSH to this box?
  - Might spit out chunks of memory once depleted

Frameworks help avoid these problems

Drip, drip, drip… the `RANGE` header attack

- Usually we use scripts/logic to protect access to resources
- But what about resumed downloads?
  - We tend to think they're safe because they've been started before
  - And therefore presumably authorised
- But what if we haven't previously authorised?
- Using a `RANGE` header an attacker could drip out the contents of a protected resource a byte at a time!
  - Must defend by forcing authentication even for resumed downloads

Less is best

- Bugs hide in code
- Bugs cause security breaches
- Less code, less places for bugs to hide
- So build only what you *need*
  - And keep your software *soft* so you can grow it over time

Defend in Depth

- Use firewalls
  - On the network, and on the server
- Do open ports 80 and 443
  - Do not open other ports
- Do not mistake HTTPS for security
  - It's not enough!
- Run at least privilege
  - Never run your service as *root* or *administrator*
- Keep good deployment hygiene
  - No lingering artifacts that attackers can grab hold of
    - Deploy only the config files, DBs, etc that you need
    - And remove what you don't
- And remember that social engineering is still effective!

Summary

- HTTP has good built-in security with HTTPS and strong cyphers/hashes
- OAuth is useful for on-behalf-of and hypermedia-tastic
- OAuth 2.0 is now more commonplace but requires transport security
- Defending is an ongoing task, not a one-off